Extracted from:

Swift Style, Second Edition

An Opinionated Guide to an Opinionated Language

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Swift Style Second Edition

An Opinionated Guide to an Opinionated Language

Erica Sadun

edited by Brian MacDonald

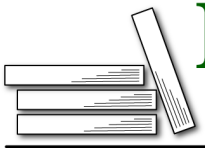
Swift Style, Second Edition

An Opinionated Guide to an Opinionated Language

Erica Sadun

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Brian MacDonald
Copy Editor: Nicole Abramowitz

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-627-3
Book version: P1.0—March 2019

This book is dedicated to the Swift community
both inside and outside Apple.

Welcome to *Swift Style, Second Edition*

This book offers a practical, powerful, and opinionated guide to coding style. It incorporates a multitude of best practices, guiding you to work successfully in this equally opinionated programming language.

This book's central theme is this: it's not just the compiler that needs to read your code. Code is more often read than written. Its crafting involves a process of review and clarification apart from developing algorithms and ensuring programmatic correctness. The better you style your code, the easier it becomes to understand and maintain. Well-structured code is an essential investment in future-proofing your work.

Since this book was first published, Swift hasn't stood still. This second edition retains the advice and insight from the first. I have revised examples to showcase an evolved Swift language, and some topics have expanded to accommodate Swift's ever-growing feature list. These pages offer you a treatise on style where the quaintness of some examples won't get in the way of your reading. This edition offers a broader discussion that follows the shape of the updated language.

Swift changed a little, but its style and this book's philosophy have not.

How This Book Got Here

This book didn't start out as a book. It began its life as a blog post ("Swift: Don't do that"[1]) and then as a GitHub repository.[2] I was inspired by one of Natasha Murashev's posts about the new open source SwiftLint[3] project, developed by JP Simard[4] and the folks at Realm.[5] At the time, Murashev wrote

1. http://ericasadun.com/2015/05/05/swift-dont-do-that/
2. https://github.com/erica/testlint
3. https://github.com/realm/SwiftLint
4. https://github.com/jpsim
5. https://github.com/realm

a weekly curated Swift newsletter that covered news and articles about the language.

At that time, SwiftLint was built on what can be kindly described as extreme hacking by hooking into Swift's then-private SourceKit services. As I wrote,[6] I loved the idea behind SwiftLint, but I needed a better tool for immediate use. I wanted something good enough for real code, even in imperfect form. It had to run from the command line and work with playgrounds. It had to be able to scan files added to Xcode projects by reference as well as those physically stored in the project root. Leveraging some of the code I had developed for a much earlier `Xcode Manifest` application, I built an Objective-C scanner[7] with simple matching rules based on regular expressions.

My linter worked line by line instead of parsing because I didn't want to use interprocess hacking. (Its implementation relied heavily on regular expression matching, waving my hands, and sacrificing chicken entrails.) It wasn't a great linter, but it was an amazing way to start thinking about the way I do and should write Swift. Over time, my rule base grew and grew as my interest evolved from "How do I automate simple code checks?" to "What are the right things to check in my code?"

I kept working to incorporate rules that I found valuable. The project quickly evolved from a project about linting into a way I could *explore* style rules for this new language. As I developed my style preferences, I found myself talking to Swift developers, asking for opinions and feedback, and incorporating their experiences into my writing.

My style sheet kept growing, maturing from a few pages of directives to a pamphlet to a small book. At a certain point, I realized I had something of more general utility than a personal style guide on my hands. And that was when I approached The Pragmatic Bookshelf about the possibility of transforming my material into a proper book: the book you are now reading.

From the time I started work on this project (way back in Swift 1!) to today, Swift settled down. It decided exactly what kind of language it was going to be, and it has finally realized and expressed that language in a much more stable form. A significant developer base has adopted the language, and these developers now have several years of Swift development experience behind them. Swift moved from the turbulence between Swift 2 and Swift 3 into calmer waters, introducing primarily additive features in Swift 4 and Swift 5.

6. http://ericasadun.com/2015/05/18/swift-alternative-lintage/
7. https://github.com/erica/testlint

The language became stronger and more reliable. It found its core philosophy and developed a firm opinion of what it was and what it wanted to be.

Today's Swift opens itself to evaluation, curation, and guidance. It allows you to weigh preferred coding styles even though the community as a whole has not yet adopted fixed or even universal conventions. You can finally build house guidelines without having the rug pulled out from under you every few months. Swift has *arrived*.

This book is written for Swift coders both new and experienced. The material in these pages ranges from "Captain Obvious" good practices to power strategies, spanning the full range between those points. I solicited opinions and preferences along the way, taking into account varied experience, programming domains, and conventions, to build a strong base of practical and practicing advice.

There are always ways to enhance your code and coding practices. While code can have issues, those issues should be evaluated with respect to strong design principles, not odor. Smell the roses, not the code, and embrace your personal style.

What's in This Book

Apple's Swift programming language has finally reached a level of stability that opens its use to a much wider audience. *Swift Style, Second Edition* guides you through the ins and outs of at least part of Swift programming's best practices. This book is written for Swift programmers both experienced and new to this language who want to explore the art of crafting code in this language.

Code style matters. Critical dos and don'ts for writing readable Swift code guide you to better craftsmanship. This book explores common coding challenges and the best practices that address them. From spacing, bracing, and semicolons to proper API style, discover both *what* you should do and the *whys* behind each recommendation.

A style guide establishes a consistent experience of well-crafted code that lets you focus on the code's underlying meaning, intent, and implementation. This book explores the areas of Swift where coding structure comes into play. Whether you're developing a personal style or a house style, there are always ways to enhance your code choices. You'll find here the ideas and principles to establish or evolve your own best style practices.

Begin with simple syntactical styling. Strengthen code bracing for easy readability. Style your closures for safety and resilience. Perfect your spacing and layout. Master literal initialization and typing. Optimize control flow layout and improve conditional style choices. Transition from Objective-C and move code into Swift the right way. Improve API design using proper naming and labeling. Elevate defaulted arguments and variadics to their right places. *Swift Style, Second Edition* covers it all.

Here's a chapter-by-chapter breakdown of what you'll find in this book:

- Chapter 1, Adopt Conventional Styling, on page ?: Conventional style is like a window. It allows you to look through to see the code intent or landscape that lies beyond it. When there's a streak on the window or an unconventional use in code, programmers naturally fixate on the wrong thing. This chapter explores common house-style elements. It surveys Swift language features you'll want to standardize and lock down for better and more consistent code.
- Chapter 2, Structure Your Code for Readability, on page ?: Swift code faces wrapping challenges you don't encounter in C-like languages. Swift's top-heavy style of output magnifies the importance of braces, wrapping, and line composition. This chapter explores ways you choose to build coding backbones. You'll learn to enhance code readability and emphasize code meaning and design intent.
- Chapter 3, Establish Preferred Practices, on page ?: Like any language, there's always more than one way to achieve your goals in Swift. Making good choices is a critical part of coding style. This chapter surveys typical design decision points and guides you through refactoring opportunities. Step beyond simple linting to explore the architectural points that affect your development.
- Chapter 4, Design the Right APIs, on page ?: An API establishes a contract for calling code. It describes the types, methods, and results produced by an implementation and how these will behave. A well-designed API provides a clear and understandable set of tools, with well-chosen names and thoughtful consideration to resilience and long-term code evolution. Learn the "Swifty" ways to design APIs by leveraging the principles of clarity, concision, and utility. From access control to naming, from nesting to defaults, this chapter shows you how to present your functionality safely and meaningfully.
- Chapter 5, Look to the Past and the Future, on page ?: Time plays a key role in Swift style. Seeing Swift in this larger scope is important in both

adopting new practices and supporting code beyond its moment of creation. This short chapter explores time, guiding you into routines that enhance, document, and support the Swift code you're now writing. You'll learn ways you can best move on from your code's past, and discover how you will support its future.

- Chapter 6, Good Code, on page ?: This book wraps up with a short meditation about what good code *means* and how you can recognize what good coding *is*.

Contributing to This Book

When reading through this volume, if you see something that's missing or that's wrong, or if you just want to suggest an alternative viewpoint, drop me a note at erica@ericasadun.com and I will consider it for a future update. Thank you in advance for being part of this effort.

Online Resources

This book has its own web page,[8] where you can find more information about the book, and you can help improve the book by reporting errata, including content suggestions and typos.

Other valuable online resources include these:

Apple's Swift resources page[9] provides links to essential Swift and Xcode materials, including iTunes U courses, videos, sample code, and a link to the official Swift Programming Series books.

The Swift.org[10] website offers a central hub for the Swift programming language. It hosts a blog,[11] which provides regular updates about topics important to those who use Swift as part of their daily coding work. You'll find announcements about language releases, the tools that support language development, new language features, and more. Apple's original Swift blog[12] is no longer maintained.

8. https://pragprog.com/book/esswift2/swift-style
9. https://developer.apple.com/swift/resources/
10. https://swift.org
11. https://swift.org/blog
12. https://developer.apple.com/swift/blog/

Swift's core documentation can now be found at docs.swift.org.[13] This includes the current version of *The Swift Programming Language* and links to the latest *API Design Guidelines* and *Migration Guides*.

The Swift Programming Language is a free ebook offered by Apple on docs.swift.org, on the Apple Books store, and on its developer website.[14] It offers the authoritative Swift reference, including a guided tour to the language and a formal reference. The book is available under the Creative Commons Attribution 4.0 International License (CC BY 4.0), and volunteers are sought to translate it to other languages.

A second Apple ebook, *Using Swift with Cocoa and Objective-C*, is just as essential. This ebook[15] provides an overview of the topics related to interoperability between the two languages and the details of how API calls straddle the two. It's a much shorter volume than *The Swift Programming Language* because of its tight focus.

Apple's online Swift Standard Library Reference[16] adds an indispensable overview of Swift's base functionality layer. It offers an overview of fundamental data types, common data structures, functions, methods, and protocols. If you stop learning Swift at the language basics, you'll be missing out on this critical portion of core language expressiveness.

Join in at forums.swift.org.[17] This site offers user forums where you can discuss the language with others. Reserved discussion areas include language announcements, topics related to the Swift Evolution process, the development and implementation of Swift, user-to-user support, and projects related to Swift including SwiftLint,[18] Kitura,[19] Vapor,[20] SourceKitten,[21] and more.

Sadly, the official Apple Developer Forums[22] are underused. Most people join a variety of Slack and IRC discussions for peer support. The official Swift user forums are an excellent resource. The conversation, at least at this time, tends to be thoughtful and the traffic is light, with good access to high-level Swift

13. https://swift.org/documentation/
14. https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language
15. https://developer.apple.com/library/content/documentation/Swift/Conceptual/BuildingCocoaApps/index.html
16. https://developer.apple.com/library/ios/documentation/General/Reference/SwiftStandardLibraryReference/index.html
17. https://forums.swift.org
18. https://github.com/realm/SwiftLint
19. https://www.kitura.io
20. https://github.com/vapor/vapor
21. https://github.com/jpsim/SourceKitten
22. https://forums.developer.apple.com/community/xcode/swift

developers. As with any language development forum, this one is meant to discuss language features with other users and is not for debugging support.

File your bugs at bugs.swift.org.[23] This open bug-reporting system enables you to submit bugs, track them, search for topics, and more. There's no need for workarounds like Tim Burks' Open Radar (site[24], repo[25]) to struggle against Apple's internal opaqueness. Each bug is visible, readable, and trackable. Submitting a bug report for Swift is completely different from Apple's black hole. You can enact change that you'll be able to see and follow.

Track Swift's progress toward ABI stability at the swift.org ABI Dashboard.[26] The Swift ABI Stability Manifesto[27] describes the tasks needed to complete the process for ABI stability, and the dashboard lists the tasks, tracking issues, and status of this ongoing project.

The swift.org APIs project also has its own page,[28] supporting Swift for server-side development. Core capabilities for networking and security allow the creation of frameworks and server applications using pure Swift code.

The core Swift repository[29] provides public access to the Swift language. Anyone with an interest to do so can download a copy of the Swift source code and build the Swift compiler for a variety of platforms. If you're so inclined, you can make changes to the Swift source, whether bug fixes or enhancements, and submit pull requests with those changes.

If you'd like to keep on top of the language and its changes, I post regular updates about Swift and developer tools on Twitter[30] and at my personal website.[31]

Language Evolution

When Swift first went open source, developers were invited to take part in bug fixes and language design. Over 200 people outside of Apple began to work on the Swift programming language and contributed to the first open

23. https://bugs.swift.org
24. http://openradar.appspot.com
25. https://github.com/timburks/openradar
26. https://swift.org/abi-stability/
27. https://github.com/apple/swift/blob/master/docs/ABIStabilityManifesto.md
28. https://swift.org/server-apis/
29. https://github.com/apple/swift
30. http://twitter.com/ericasadun
31. http://ericasadun.com

source release, Swift 2.2, in March 2016. The number of Swift contributors is now in the mid to high hundreds.

Although many initial changes were little more than typo corrections (essentially low-hanging fruit), the scope and sophistication of external contributions has grown. Many new Apple employees started as open source contributors. If you're interested in participating in Swift language design, they use a strict internal style, which is touched on in this book.

The Swift Evolution repository[32] hosts an archive of proposals affecting user-visible enhancements for the Swift programming language. User-facing changes must proceed through the evolution process, which is chaired by a core team of language experts before being adopted into the language.

Its associated github.io page[33] provides an easy-to-consume list of proposals, including those in active review, those that have been accepted, and more. This Swift Evolution overview page goes in reverse order, with newer proposals listed at the top. This is where you find items that are just now being considered.

As you scroll down, you'll find accepted proposals, proposals that have been returned for revision, and deferred, withdrawn, and rejected proposals. I hold the current record for most-rejected proposals for the Swift programming language. And I still think that adding trailing commas in parameter lists and tuples is an excellent enhancement, especially since the language supports defaulted arguments.

Some proposals have been accepted but not implemented, such as SE-0068,[34] which expands the use of uppercase `Self` to class members and value types, as you would use it in Objective-C. It is proposals like this one that led Swift Evolution to adopt a rule that you must provide a working implementation *before* a proposal can be reviewed. Just because a proposal is a great idea does not mean that it is implementable.

Swift's change log[35] offers details about what has changed in each release, including releases that have not yet been distributed through Xcode. If you'd like to peek at upcoming features, the log is updated as each new feature is implemented and added to the master language branch.

32. https://github.com/apple/swift-evolution
33. https://apple.github.io/swift-evolution/
34. https://github.com/apple/swift-evolution/blob/master/proposals/0068-universal-self.md
35. https://github.com/apple/swift/blob/master/CHANGELOG.md

Any change log entry that starts with the letters SR means that it is responding to a bug report. Any entry listed with SE indicates a Swift Evolution proposal that has been adopted and implemented into the language. Like the status page, the changes scroll from most recent to least, and go back further than the open source epoch. It's fascinating to delve into the language's design decisions that predate December 2015, when Swift was opened up to the world.

New language versions that have not been released publicly can be downloaded via a system of nightly releases. Visit the swift.org downloads page.[36] You'll find both current installers for macOS and Ubuntu Linux, along with development builds. Although official support for installers are limited to Mac and Linux, Swift has been ported to Windows' Linux subsystem, Android, FreeBSD, Raspberry Pi, and more.

Ready to Get Going?

You're about to take a deep dive into Swift style. This book explores the fine details of language use in a fascinatingly pedantic (and possibly diagnostically obsessive) fashion. This book won't change your understanding of Swift and it won't reduce the pain of writing and debugging your code, but it should take a lot of the pain out of reading and reviewing that code at some later date. This book offers new ways to think about Swift, to structure Swift, and to make good coding choices.

Thank you for purchasing this book. Let's get started!

Credits

Thanks

My thanks to everyone who helped provide technical information and feedback during the development of this book. I'm deeply indebted to my technical

36. https://swift.org/download/

review team: Lily Ballard(!), Tony Gray, Maurice Kelly, Stephen Wolff, Stefan Turalski, Peter Wood, Nick Watts, Alex Henry, Arun Kumar, Daivid Morgan, Derek Graham, Dianne Siebold, Gábor László Hajba, Kenneth Geisshirt, Nick McGinness, Peter Hampton, and Phillip Apley. Additional shout-outs go to Steve Streza, Mike Ash, Alex Kempgen, Seth Willets, Gwynne Raskind, Xiaodi Wu, Kenny Leung, Paul Cantrell, Sebastian Celis, Jeremy Tregunna, Nate Cook, Pyry Jahkola, Russ Bishop, Jacob Bandes-Storch, Soroush Khanlou, Mars Geldard, Tim Vermeulen, Ian Mateus Vieira Manor, Matthew Johnson, Andrew Bennett, Kevin Cramer, Adam Sharp, Connor Neville, Anthony Mattox, Alfonso Urdaneta, Brent Royal-Gordon, Zev Eisenberg, Mohamed Fouad, Cezary Wojcik, Seivan Heidari, Jessy and Catie Catterwaul, Dave DeLong, Rainer Brockerhoff, Brian King, Peter Livesey, Joe Fabisevich, Mark Rowe, Grant Isom, Olivier Halligon, Tim Nugent, Jeff Forbes, and Greg Titus. Thanks go out to the Swift Evolution and Swift Users' Group members and everyone at Apple who has provided inspiration, support, and insight, including but not limited to Chris Lattner, Joe Groff, Ted Kremenek, John McCall, David Smith, Doug Gregor, Dave Abrahams, Jack Lawrence, Brian Gesiak, and Jordan Rose.

Thanks also to my Pragmatic Team: Brian MacDonald, development editor and all-around great guy who managed and oversaw with patience, insight, and humor; Andy Hunt, publisher; and everyone else at The Pragmatic Bookshelf who helped make this book a reality.

A final, special, thank you to Allen Denison, who helped make it possible for the slug on the cover of the second edition to hold a coffee cup with an official Swift logo! It's a tiny touch that took a lot of work to make happen and I'm so grateful for his perseverance through so many levels of bureaucracy to support me.