BJ Miller

Sams Teach Yourself

Swift

in 24 Hours

SAMS

BJ Miller

Sams Teach Yourself Swift in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

ISBN-13: 978-0-672-33724-6

ISBN-10: 0-672-33724-X

Library of Congress Control Number: 2014948978

Printed in the United States of America

Second Printing February 2015

Editor-in-Chief
Mark Taub

Senior Acquisitions Editor
Trina MacDonald

Senior Development Editor
Chris Zahn

Managing Editor
Kristy Hart

Senior Project Editor
Betsy Gratner

Copy Editor
Geneil Breeze

Indexer
Lisa Stumpf

Proofreader
Leslie Joseph

Technical Editors
Valerie Shipbaugh
Hank Turowski

Editorial Assistant
Olivia Basegio

Cover Designer
Mark Shirar

Compositor
Nonie Ratcliff

Contents at a Glance

Table of Contents

About the Author

BJ Miller is an iOS developer for a consultancy in the Cleveland, Ohio, area. BJ earned his B.S. in Computer Science from Baldwin-Wallace College (now called Baldwin-Wallace University) in Berea, Ohio, the town where he grew up. His latest career path encompasses large-scale enterprise network administration, SQL database administration, and Microsoft SharePoint Server and Microsoft Project Server administration and integration as a contractor for the United States Department of Defense, with all the Microsoft certifications that come along with that. Before that, he spent several years in LAN engineering, designing and implementing network infrastructure, as a Cisco Certified Network Associate.

BJ began iOS development in 2009 after not having programmed for a few years, and he developed a passion for the platform and the Objective-C language. Now, his love has expanded to include Swift, and there is still yet room in his heart for more. In 2013 he released his first app into the iOS App Store, called MyPrayerMap, as a simple tool for managing prayer requests.

When he is not writing in Objective-C or Swift for either work or this book, he enjoys spending time with his wife and two boys, reading, listening to music or podcasts, and playing The Legend of Zelda (any game on any system will do). He also co-organizes the Cleveland CocoaHeads Meetup with Daniel Steinberg, http://www.meetup.com/Cleveland-CocoaHeads/, and organizes a submeetup of that group called Paired Programming Fun, which is a casual meetup where the focus is on Test-Driven Development (TDD) in Swift and Objective-C in paired-programming style. BJ often presents iOS-related topics at CocoaHeads and also speaks at other conferences such as CocoaConf (Columbus, Ohio) and CodeMash v2.0.1.5. He also blogs from time to time at http://bjmiller.me and is on Twitter as @bjmillerltd.

Dedication

This book is dedicated to my wonderful family and friends who have been incredibly supportive throughout this entire process. Thank you all for your love and encouragement.

Acknowledgments

I would like to thank my wife and two boys for putting up with me while I wrote this book. I am excited to spend more time with you. I would also like to thank whoever invented coffee; may the Lord bless your soul and keep you. Speaking of the Lord, it is pretty close to not humanly possible that this book would have been completed in time, with all the other obstacles going on in my life, without his loving arms around me and my family; thank you, Jesus.

I would also like to thank my friends, coworkers, and the rest of the Mac/iOS community for all their love and encouragement. If I had not been introduced to someone, who introduced me to someone, who introduced me to Daniel Steinberg, I might not have pursued iOS development further and I might not have written this book. If you ever get the chance to meet that man, your life will be enriched.

Also, I cannot go without thanking the fine people who helped this book come to be: Trina MacDonald (Acquisitions Editor), Chris Zahn (Senior Development Editor), Olivia Basegio (Editorial Assistant), Hank Turowski (Technical Editor), Geneil Breeze (Copy Editor), and Betsy Gratner (Project Editor).

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: errata@informit.com

Mail: Sams Publishing
ATTN: Reader Feedback
330 Hudson Street
7th Floor
New York, New York 10013

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

At Apple's yearly World Wide Developer Conference (WWDC) in June 2014, Apple announced a new programming language called Swift that the company had been developing since 2010. This was a huge announcement; Objective-C had been the primary language of choice for developing most Mac and iOS apps for many years. The excitement surrounding this language was palpable. Twitter lit up with tweets about Swift, domain names with Swift in the title were being bought up left and right, and within 24 hours of the announcement more than 300,000 copies of Apple's Swift iBook had been downloaded. People were ready for change.

But a new language brings not only syntactic differences but also idiomatic differences and new conventions. Swift is not just an object-oriented language, but it introduces features gleaned from other languages, such as C#, Haskell, Ruby, and more. Touted to be "Objective-C without the C," Swift builds upon familiar concepts from Objective-C but includes a more modern, safer syntax and multiple paradigms such as object-oriented, functional, imperative, and block structured.

Swift is officially at version 1.0 but is still evolving, and even as this book is being written more changes are entering beta. With that said, this book is current as of Swift 1.0 and Xcode 6.0.1. If there are changes that you find in these examples that do not work as described or with screenshots, please check Apple's release documentation and electronic versions of this book as they can get updated a lot faster than the printed book you may have in your hands. Also, all the code examples from this book are available and will be kept up-to-date in the GitHub repository, https://github.com/STYSwiftIn24H/Examples.

Swift is already proving to be a great language and as of its release is compatible with iOS 7 and up. Swift can be written for apps running on OS X Yosemite, but as of Xcode 6.0.1, the option for adding Swift files to an Objective-C project, or even creating a Mac app using Swift are not available. More Mac support is coming in future releases. Updates are coming from Apple rather quickly, so if something is not available that you need or if something is not working as expected, consider filing a bug or feature request at http://bugreport.apple.com.

Who Should Read This Book?

This book is designed for a beginner-intermediate level programmer. Even advanced programmers who are not yet familiar with Swift can benefit from this book. You do not have to have a background in software development to make your way through this book,

although it may help. If you are not familiar with software development whatsoever, you may benefit from more fundamental books first, although with the examples inside this book you may be able to follow along just fine.

In this book, I assume you have a passion to learn about Swift and to develop apps for the Mac and/or iOS platforms. I also assume that you are willing to carve out time in your schedule to take this book seriously and learn the concepts herein.

What Should You Expect from This Book?

This book is a guided tour of the Swift programming language, discussing some of the ins-and-outs of Swift, best practices, do's-and-don'ts, and more. It is not just a language reference. By the time you complete this book you should have a firm grasp on many of the concepts in Swift including the syntax to make them come to fruition.

You should not expect to be able to write award-winning iOS or Mac apps right out of the gate by just reading this book alone, as this book is not meant to be a one-stop-shop for learning everything about app creation. Such a book would be thousands of pages long. Rather, there are more components to writing apps, particularly the Cocoa and Cocoa Touch frameworks, which deserve books in their own right (and many exist). Apps should be written with careful planning and development, and depending on how many different technologies your app includes, you may need more resources.

You also do not need to read this book from cover to cover before attempting to write apps of your own using Swift. Feel free to experiment along the way with your own apps, or use this book for reference if you are stuck in an app of your own and need some guidance.

Also remember that this book is current as of Swift 1.0 and Xcode 6.0.1, so please understand that changes may be made after this book has gone through final edits and been printed. Code examples will be updated as progressions in the Swift language and Xcode environment change. They are available on GitHub at https://github.com/STYSwiftIn24H/Examples.

NOTE

Code-Continuation Arrows and Listing Line Numbers

You'll see code-continuation arrows ➥ occasionally in this book to indicate when a line of code is too long to fit on the printed page.

Also, many listings have line numbers and some do not. The listings that have line numbers have them so that I can reference code by line; the listings that do not have line numbers are not called out by line.

HOUR 1

Introducing the Swift Development Environment

What You'll Learn in This Hour:

- What Swift is and where it came from
- How to install Xcode 6 from the Mac App Store
- How to navigate the Xcode Integrated Development Environment (IDE)
- How to use playgrounds
- How to use Swift's Read-Eval-Print-Loop (REPL)
- How to write your first Swift app

Since the introduction of the iPhone in 2007, Apple seems to have lit a fire in the industry for not only consumer-based electronics but also the opportunity for most anyone to be able to write apps for their platform, be it Mac or iOS. This has had a dramatic effect on culture, as you cannot go to a coffee shop or to any business now and not see a slew of MacBook Airs, MacBook Pros, iPhones, and iPads. Chances are, if you're reading this book, you are wondering how you can write an app that could appear on the screens of the very people you see at those coffee shops and businesses.

This book is about the Swift programming language, the new programming language announced by Apple at the 2014 World Wide Developer Conference (WWDC). Prior to Swift's introduction, Mac and iOS apps were mainly written in a language called Objective-C, which is a strict superset of the C programming language, meaning that you could write apps in both languages, and sometimes had to. In this book we explore the Swift programming language and learn its fundamentals, structure, and syntax, which gives you a foundation to write great Mac and iOS apps.

What Is Swift?

Swift is a programming language customized by Apple and introduced as Objective-C without the C. Indeed, this is true, but Swift has also taken cues from other languages such as Haskell, Ruby, Python, C#, and several others. Swift is tuned to work with the existing Cocoa and Cocoa Touch frameworks, which contain all the familiar classes used in modern Mac and iOS apps, to support their interoperability.

Swift is built on three pillars: being safe, powerful, and modern. Swift provides a lot of safety in terms of type checking, constants for immutability, requiring values to be initialized before use, built-in overflow handling, and automatic memory management. With respect to power, Swift was built using the highly optimized LLVM compiler, includes many low-level C-like functions such as primitive types and flow control, and of course was built with Apple's hardware in mind for optimal performance. Swift is also modern in that it adopted many features from other languages to make the language more concise yet expressive, such as closures, generics, tuples, functional programming patterns, and more that we cover in later hours.

Getting Started

The biggest assumption at this point is that you have a Mac computer already, as without that, you cannot install Xcode, Apple's Mac and iOS Integrated Development Environment (IDE).

NOTE

Download Xcode

Xcode 6 is a free download from the Mac App Store. You must have Mac OS X 10.9.3 or later.

Launch the App Store app on your Mac, search for Xcode, and click to install the software. Once the installation is complete, Xcode is listed in your `/Applications` directory.

Take a Look Around

When you open Xcode, you may be greeted with prompts asking whether you want to install extra tools; go ahead and install them. This should only happen the first time you launch Xcode. Once Xcode is open, you see a standard menu window with options to create a playground, create a new project, or open an existing project, and on the right side is a list of recent projects and playgrounds you've opened (if you've opened any). The window should look like Figure 1.1.

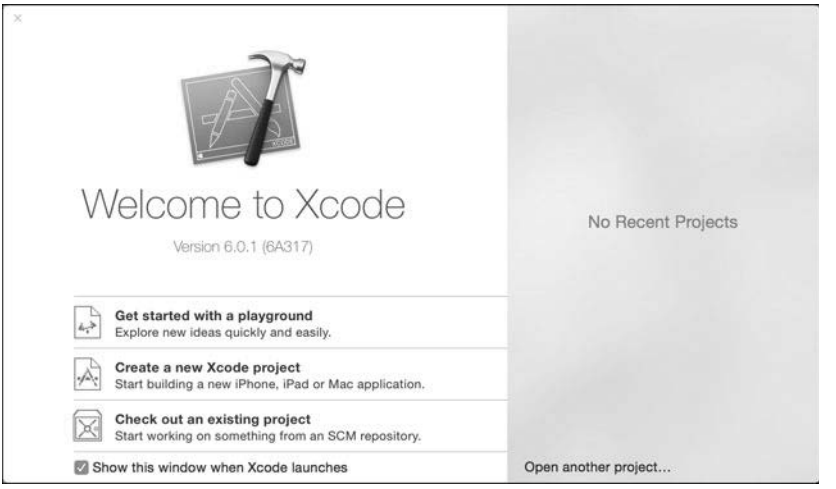

FIGURE 1.1
The Welcome to Xcode screen, where you can choose to create or edit projects and playgrounds.

Although in this book we predominantly work in playgrounds, it is good to become familiar with the IDE, so let's do that quickly. Click Create a New Xcode Project to create a new Xcode project. The next screen asks you what type of project you want to create, and for this experiment, just use Single View Application, as seen in Figure 1.2, and click Next.

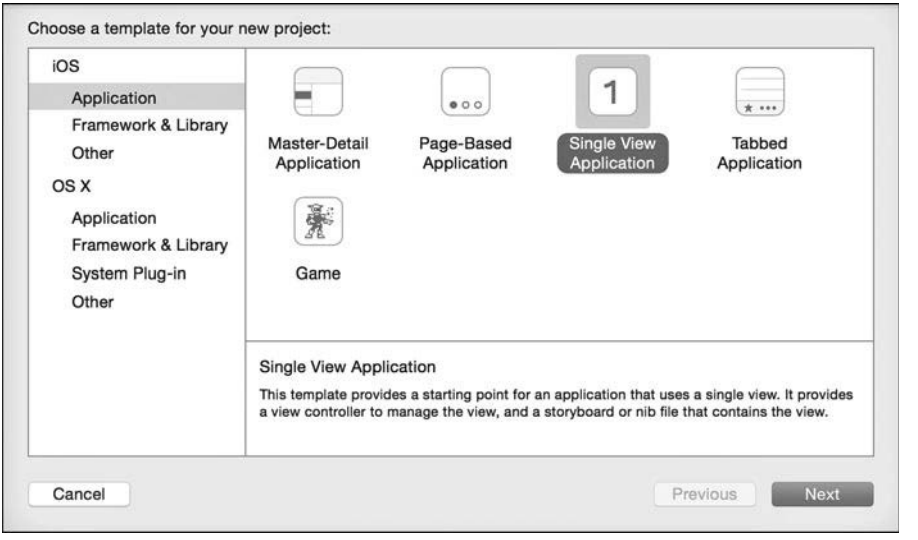

FIGURE 1.2
The project template chooser screen.

Next you are asked to name your project. Choose an Organization Name, Identifier, Language (Swift or Objective-C), and Device(s) to run on, and indicate whether you want to use Core Data, as shown in Figure 1.3. All this information is useful for future projects you will create, but for our testing purposes, we don't need to worry about it yet. The Organization Identifier is usually a reverse-DNS name of your personal or company URL to ensure uniqueness at an organizational level, and the Bundle Identifier tacks on the Project Name to the end of the Organization Identifier to ensure uniqueness per app bundle. Once you submit an app to either the Mac App Store or the iOS App Store, your bundle identifier needs to be unique.

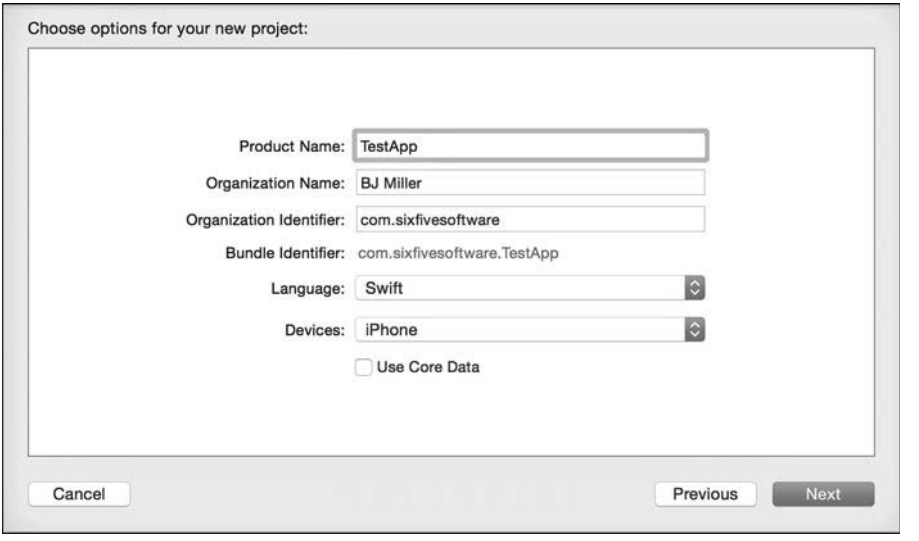

FIGURE 1.3
Enter your project-related information.

Here you also have the choice for device type, such as iPhone or iPad, and that is so that Xcode can properly create the Storyboard files needed for the device or devices you plan to write your app to be used on. This way you can target different interfaces on different devices, but still use the same code to manage them both, such as with Universal apps.

Name your project TestApp, since we just want to get to Xcode to get acclimated (you can remove this project later), choose Swift for the language, and click Next. Xcode opens, and you see the new project you created. It should look something like Figure 1.4.

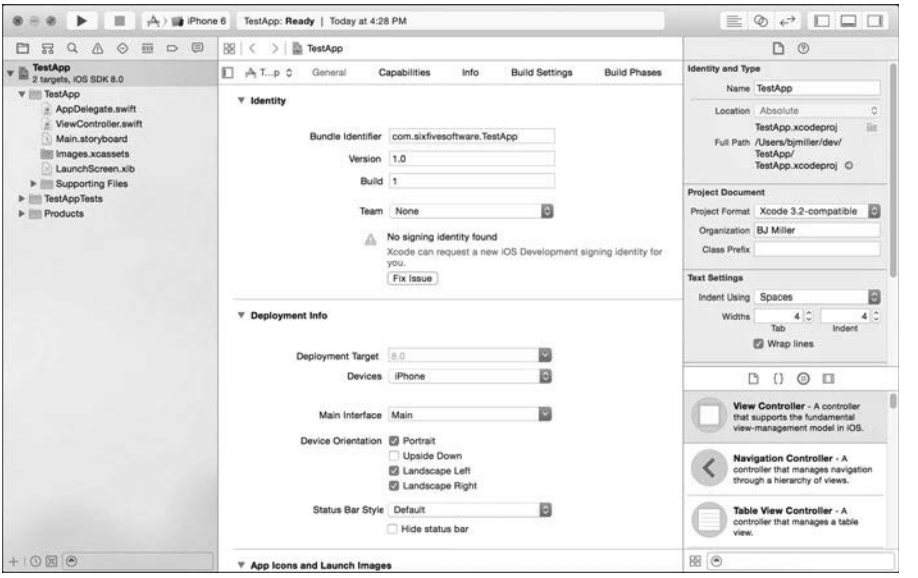

FIGURE 1.4
The initial Xcode IDE layout. The Navigation Pane is on the left, Inspector Pane is on the right, and main content area is in the center. In the top toolbar are buttons to run or stop a build, see error and warning info, and show or hide views.

Xcode is nicely partitioned off into logical sections, as you may be accustomed to from other IDEs; however, it also has some nice features to note. The pane on the left-hand side is called the Navigator Pane. Here, you can choose between different Navigators to view files in your project, warnings and errors, breakpoints, Unit Tests, and more. The pane on the right-hand side is called the Inspector Pane. This dynamic pane changes depending on what element is clicked, such as editing a selected button's text property, or adjusting a control's position in a window.

The main content area in the center is where you'll spend most of your time when working on an actual Mac/iOS project. The content area is where you can change project settings, and most importantly create your app by either writing code or designing the interface in a Storyboard.

The bar along the top left has several useful functions available. Toward the left, you have your standard Mac Red/Yellow/Green buttons for window management. Next, the play and stop buttons are actually *Build and Run* and *Stop* when referring to compiling, building, and running your apps on the Simulator or devices. To the right of that is where Xcode notifies you of current information, such as how many warnings or errors are in your project, and build status.

Finally, the upper right-hand set of buttons can adjust the views you see to show or hide the Navigator Pane, Inspector Pane, Debug Pane, or Assistant Editor, as well as view code comparisons, source code "blame" views, and logs.

NOTE

Viewing Two Files

The Assistant Editor splits the content area in half so you can view two files at the same time. This is helpful, for instance, when you might be writing Unit Tests and also the code to make the Unit Tests pass, or if you are creating a user interface in a Storyboard and have the corresponding View Controller open to connect Actions and Outlets. For more on developing apps with custom interfaces, I recommend reading John Ray's book *Sams Teach Yourself iOS Application Development in 24 Hours*, Sams Publishing.

Xcode Playgrounds

One of the excellent new features of Xcode 6 is something called **playgrounds**. A playground is a scratch pad, if you will, for testing out code to ensure you receive proper results from code segments, before adding the code to your app. It is because of this functionality that playgrounds are so powerful; you can get immediate feedback if your code is going to give you the results you expect without having to compile your code and run it on the Simulator or a device.

You can create a new playground at any time, and you can choose to have it be a part of your project or just as an independent playground file. Since we're already in an open project, click File > New > File, and then in the Source set of files (for either iOS or Mac), choose playground, then click Next. In the Save As dialog, name your playground file (the name MyPlayground is fine for our purposes), and then click Create. Don't worry about the Group or the Targets for now, we aren't building an app yet so we don't care about that.

NOTE

Mac or iOS Playgrounds

There is a difference in file structure when creating a playground from the Mac section or iOS section. Creating a Mac playground adds the `import Cocoa` statement at the top of the file, and makes Mac frameworks and modules available to you. Creating an iOS playground adds the `import UIKit` statement at the top of the file, and makes iOS framekworks and modules available to you. There is nothing visually different about either playground at first. If you create an iOS playground, but instead want to test Mac app code, or vice-versa, simply create a new playground of the desired type.

Notice that your new playground comes equipped with a few lines of Swift code for your learning convenience. Let's touch on a few of these basics first before moving on. Your playground should look something like this:

```
// Playground - noun: a place where people can play
import UIKit
var str = "Hello, playground"
```

The first line in the preceding code is a comment and is ignored by the compiler. You can use comments to annotate certain parts of code to be human readable, perhaps explaining for other coworkers (or even yourself) what this particular section of code is for. The `//` (double forward slash) signifies that the remainder of the line is to be treated as a comment. You can also comment entire sections of code or paragraphs of text, either on the same line or on multiple consecutive lines by enclosing them in `/*` and `*/`. Swift even allows you to nest comment blocks inside comment blocks, such as `/* ... /* ... */ ... */`.

The remainder of the preceding code performs a simple task in that it assigns the string "Hello, playground" to a variable `str`. Even though the code doesn't directly print any output, the playground by default displays "Hello, playground" in the playground's results pane to show the contents of the variable and any subsequent variables or constants you create. This comes in handy when you want to test logic, math, and other operations.

TRY IT YOURSELF ▼

Create Your First Lines of Swift in the Playground

At this point, it makes sense to try out your first lines of code while you're in the playground, so let's do it together here.

1. Type the following onto a new line in the playground:

```
let myNewValue = 40 + 2
```

2. Notice the right-hand side of the playground displays "`42`". Type the following line of code as-is, to insert the value inside a sentence.

```
println("My new value is \(myNewValue).")
```

Congratulations! You have written your very first lines of Swift.

In the preceding Try It Yourself example, you assigned a value of 42 to `myNewValue` in Step 1. Then, in Step 2 you inserted the value inside a sentence, using something called **string interpolation**, which is a convenient way to interpolate variables or constants inside output. The next hour discusses string interpolation in more detail. The `println()` statement prints output to the console, which is handy for quick debugging or viewing contents of data.

The Swift Read-Eval-Print-Loop (REPL)

Swift also comes packaged with a nice feature called a **Read-Eval-Print-Loop**, or a **REPL** for short. The REPL is an interactive command-line based version of what we just experienced with playgrounds. Using the REPL is nice for quick tests to make sure code works the way you expect, similar to what you get with a playground, but rather than creating a new file in your project,

you can just use this ephemeral REPL to get in, test your code, and get out. Whether to use a playground or the REPL is largely a matter of preference of what you feel comfortable with. If you are already using Terminal.app, or some other command-line utility, it may be easier for you to just open the REPL. On the other hand, if you're already in Xcode, it may be quicker for you to just create a playground and go from there.

To access the REPL, you simply type the following:

```
$> xcrun swift
```

`xcrun` is a command-line tool provided by Xcode for running or locating development tools or properties. So in the preceding line, we're telling `xcrun` to run the Swift REPL. When you press the Return key, you see the following:

```
Welcome to Swift!  Type :help for assistance.
  1>
```

The `1>` is the Swift REPL prompt where you can start typing Swift code, one instruction per line, and it interprets your code for you, much like the playground did. Let's try another example of writing code, this time in the Swift REPL.

▼ TRY IT YOURSELF

Combine Two Strings Together

Let's do another example of some Swift code here; hopefully this one isn't too difficult yet. If you don't fully understand it, don't worry; we cover this in great detail in the next hour.

1. Open Terminal.app on your Mac.
2. Type `xcrun swift` in the Terminal; then press Return.
3. At the `1>` prompt, enter the following:

   ```
   let firstHalf = "I'm writing Swift code,"
   let secondHalf = " and I'm so excited!"
   let combined = firstHalf + secondHalf
   ```

4. Notice how each time you press Return, Swift's REPL displays the name of the constant or variable we used, its data type of `String` (we cover data types in Hour 2, "Learning Swift's Fundamental Data Types"), and its value.
5. Take a look at how using the + operator concatenates the two strings together. Swift is smart enough to know that even though we're dealing with letter characters (as opposed to adding numbers), the + operator adds `String` instances together. More on operators in Hour 3, "Using Operators in Swift."

You're doing great! The Swift REPL keeps constants and variables (as well as Classes, Structs, Enums, and others) in memory for the duration of your REPL session. This means that you can reference variables, constants, classes, and so on, several lines later, which helps you work on tackling problems quickly and easily before you write the code in your actual app. The completed Try It Yourself example should look like Figure 1.5.

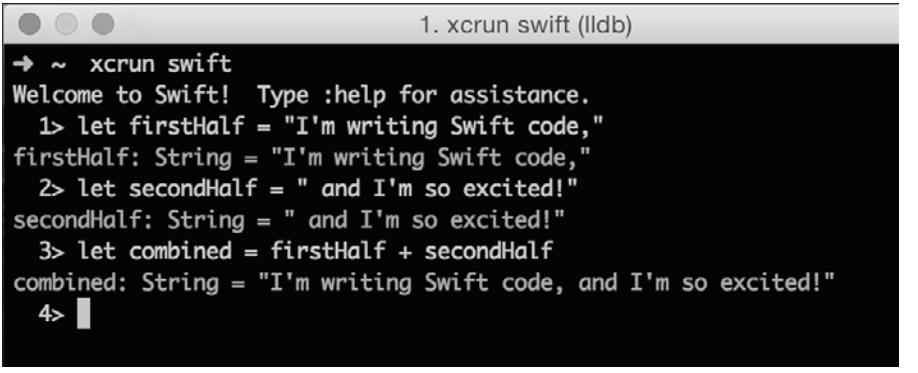

FIGURE 1.5
The completed Try It Yourself example using the Swift REPL.

To quit the Swift REPL, type a colon (:) to invoke command mode; then type q for quit and press Return. You are returned to your regular Unix shell prompt.

Summary

In this first hour, you learned a brief background on the Swift programming language and what it is built upon. We walked through opening the Xcode environment for the first time and explored some of Xcode's layout, as well as the Swift REPL. You also created your first lines of Swift code and saw how Xcode and the REPL give you instant feedback on what your code is doing.

In the next hour, we cover the difference between variables and constants, and explore some of Swift's native data types, such as `String`, `Int`, `Bool`, `Character`, `Double`, and `Float`.

Q&A

Q. Can I have a playground without having to create a full Xcode project?

A. Absolutely. Xcode treats playgrounds as interpretable files, independent of any project.

Q. I am still running OS X Mountain Lion, can I still use Xcode?

A. OS X Mountain Lion (v. 10.8) can run Xcode, but the latest version of Xcode that can run on Mountain Lion is Xcode 5.x. Without Xcode 6, you cannot use Swift.

Q. I just started learning iOS development in Objective-C, but now Apple announced the Swift programming language. Should I still learn Objective-C?

A. Objective-C is still heavily used in many Mac and iOS apps and will still be used for some time to come. This book predominantly teaches the Swift programming language, but if you want to be a developer full-time or in some sort of capacity, you may encounter Objective-C code from an existing code base, so you may benefit from learning some Objective-C. A great introduction to iOS programming in Objective-C is John Ray's book *Sams Teach Yourself iOS Application Development in 24 Hours* from Sams Publishing.

Workshop

The workshop contains quiz questions and exercises to help you solidify your understanding of the material covered. Try to answer all questions before looking at the answers that follow.

Quiz

1. What command opens the Swift REPL?
2. Use a playground to write Swift code that multiplies the numbers 3 and 19 and stores the value in a variable named `result`. What does the code look like?
3. How do you quit the Swift REPL?
4. What is the minimum Mac OS X version that runs Xcode 6?
5. What would be the output of the following Swift code?

```
let age = 33
let outputString = "Someone you know is \(age) years old"
```

Answers

1. `xcrun swift`.
2. `var result = 3 * 19` (The playground result pane displays 57.)
3. Type a colon (:) then q, and then press Return.

4. Mac OS X 10.9.3 is the minimum version to run Xcode 6.
5. The output would be: “Someone you know is 33 years old”.

Exercise

Try creating a playground or use the Swift REPL to combine two strings together, and then use the `lowercaseString` method on your combined string to convert the string to all lowercase letters. (HINT: Import Cocoa first. In a playground, press the period key (.) immediately after typing the combined variable name to get a list of actions you can take on that string.)

This page intentionally left blank

This page intentionally left blank

Index

Symbols and Numbers

A

B

C

D

E

F

G

H

I

J-K

L

M

N

O

P

Q

R

S

T

U

V

W

X-Y-Z