

BJ Miller

Sams **Teach Yourself**

Swift

in **24**
Hours



SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



BJ Miller

Sams **Teach Yourself**

Swift

in **24**
Hours

SAMS

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

Copyright © 2015 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

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

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs accompanying it.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

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

Composer

Nonie Ratcliff

Contents at a Glance

Introduction	xv
HOURL 1 Introducing the Swift Development Environment	1
2 Learning Swift’s Fundamental Data Types	13
3 Using Operators in Swift	27
4 Working with Collection Types (Arrays and Dictionaries)	39
5 Understanding Optional Values	53
6 Controlling Program Flow with Conditionals	65
7 Iterating Code with Loops	81
8 Using Functions to Perform Actions	99
9 Understanding Higher Order Functions and Closures	117
10 Learning About Structs and Classes	137
11 Implementing Class Inheritance	155
12 Harnessing the Power of Enums	171
13 Customizing Initializers of Classes, Structs, and Enums	185
14 Digging Deeper with Properties	205
15 Adding Advanced Type Functionality	223
16 Understanding Memory Allocation and References	243
17 Using Protocols to Define Behavior	263
18 Using Extensions to Add Type Functionality	285
19 Working with Optional Chaining	301
20 Introducing Generics	313
21 Adding Interoperability with Objective-C	329
22 Interacting with User Interfaces	351
23 Asynchronous Programming in Swift	373
24 Learning Swift’s Standard Library Functions	393
Index	409

Table of Contents

Introduction	xv
HOURL 1: Introducing the Swift Development Environment	1
What Is Swift?	2
Getting Started	2
Summary	9
Q&A	10
Workshop	10
Exercise	11
HOURL 2: Learning Swift's Fundamental Data Types	13
Constants in Swift	13
Variables in Swift	14
Introducing Data Types	15
Summary	23
Q&A	24
Workshop	24
Exercise	25
HOURL 3: Using Operators in Swift	27
Unary Operators	27
Binary Operators	29
Ternary Conditional Operators	35
Summary	35
Q&A	36
Workshop	36
Exercise	37

HOOR 4: Working with Collection Types	39
Arrays	39
Dictionaries	46
Tuples	50
Summary	51
Q&A	51
Workshop	52
Exercise	52
HOOR 5: Understanding Optional Values	53
What Are Optional Values?	53
How to Designate a Variable as Optional	54
Wrapping and Unwrapping Optional Variables	54
Use Case for Optionals	58
Summary	61
Q&A	62
Workshop	62
Exercise	63
HOOR 6: Controlling Program Flow with Conditionals	65
The <code>if</code> Statement	65
The <code>switch</code> Statement	70
Summary	77
Q&A	78
Workshop	78
Exercise	80
HOOR 7: Iterating Code with Loops	81
Two Categories of Loops	81
Transferring Control in Loops	93
Summary	95
Q&A	95
Workshop	96
Exercise	97

HOUR 8: Using Functions to Perform Actions	99
The Nature of Functions in Swift	100
General Function Syntax and Structure	100
Functions with No Parameters and No Return Type	102
The Type of a Function	102
Functions with Parameters	103
Functions with Variadic Parameters	105
Functions with Return Types	106
External Parameter Names	111
Default Parameter Values	112
Change Argument Values with In-Out Parameters	112
Summary	113
Q&A	114
Workshop	114
Exercise	115
HOUR 9: Understanding Higher Order Functions and Closures	117
Higher Order Functions	117
Closures	124
Summary	132
Q&A	133
Workshop	133
Exercise	135
HOUR 10: Learning About Structs and Classes	137
Overview of Structs and Classes in Swift	137
What Swift Structs and Classes Have in Common	139
Differences Between Structs and Classes	147
When to Use a Class or a Struct	150
Summary	151
Q&A	151
Workshop	152
Exercise	153

HOUR 11: Implementing Class Inheritance	155
What Is Inheritance?	155
Identifying a Base Class	156
Creating a Subclass	157
Overriding Inherited Methods	159
Accessing <code>super</code>	162
Preventing Overrides	164
Class Identity	166
Summary	167
Q&A	168
Workshop	169
Exercise	170
HOUR 12: Harnessing the Power of Enums	171
Understanding Swift Enums	171
Swift Enum Structure	172
Raw Values	173
Enum Shorthand Syntax	175
Associated Values	177
Switching Enum Values	178
Adding Instance Methods to Enums	180
Summary	182
Q&A	183
Workshop	183
Exercise	184
HOUR 13: Customizing Initializers of Classes, Structs, and Enums	185
Initialization	185
Initializing Value Types	187
Advanced Initialization	193
Summary	201
Q&A	202
Workshop	202
Exercise	203

HOOR 14: Digging Deeper with Properties	205
Stored Properties	206
Computed Properties	209
Property Accessors	209
Property Observers	212
Inheriting and Overriding Accessors	215
Inheriting and Overriding Observers	217
Summary	221
Q&A	221
Workshop	222
Exercise	222
HOOR 15: Adding Advanced Type Functionality	223
Type Properties and Methods	223
Type Aliasing	229
Type Access Control	230
Subscripts	231
Type Casting and Non-Specific Types	235
Summary	239
Q&A	239
Workshop	240
Exercise	241
HOOR 16: Understanding Memory Allocation and References	243
Deinitialization	243
Automatic Reference Counting	247
Summary	259
Q&A	260
Workshop	261
Exercise	261
HOOR 17: Using Protocols to Define Behavior	263
Defining Protocols	263
Creating and Adopting Protocols	264
Properties	265

Defining Methods in Protocols	266
Using Protocol Names as Types	268
Adopting and Inheriting Multiple Protocols	269
Optional Protocol Properties and Methods	273
How to Check for Protocol Conformance	275
Using Protocols for Delegation	275
Summary	280
Q&A	281
Workshop	282
Exercise	282
HOOR 18: Using Extensions to Add Type Functionality	285
Defining Extensions	285
Adding Functionality with Extensions	287
Summary	297
Q&A	297
Workshop	298
Exercise	298
HOOR 19: Working with Optional Chaining	301
Defining Optional Chaining	301
Chaining Optional Properties	302
Subscripts	303
Methods	308
Summary	310
Q&A	311
Workshop	311
Exercise	312
HOOR 20: Introducing Generics	313
An Introduction to Generics	313
Type Parameters and Placeholder Types	314
Specifying Type Constraints	315
Creating Generic Types	319
Extending Generic Types	322

Using Associated Types in Protocols	323
Summary	325
Q&A	326
Workshop	326
Exercise	327
HOURL 21: Adding Interoperability with Objective-C	329
Objective-C Basics	329
Bridging	335
Integrating Swift into an Objective-C App	337
Summary	346
Q&A	347
Workshop	348
Exercise	348
HOURL 22: Interacting with User Interfaces	351
Interface Introduction	351
Interface Definitions	352
Building an Interface	353
Creating Your Own Project	355
Adding UI Elements	355
Summary	369
Q&A	370
Workshop	370
Exercise	371
HOURL 23: Asynchronous Programming in Swift	373
The Problem That Concurrency Solves	373
Different Types of Queues	374
Asynchronous Programming in Action	376
Add iTunes Search to Songs App	380
Parsing JSON	383
Summary	390
Q&A	390
Workshop	391
Exercise	392

HOOR 24: Learning Swift's Standard Library Functions	393
What Is a Standard Library?	393
Numeric Types	394
String Type	394
Protocols	395
Functional Functions	399
Global Functions	401
Summary	404
Q&A	405
Workshop	405
Exercise	406
Index	409

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.

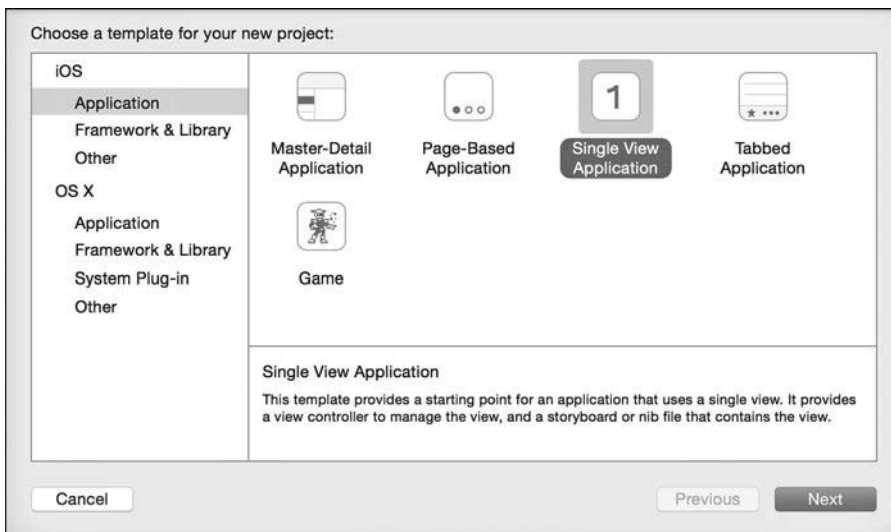
Take a Look Around

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**

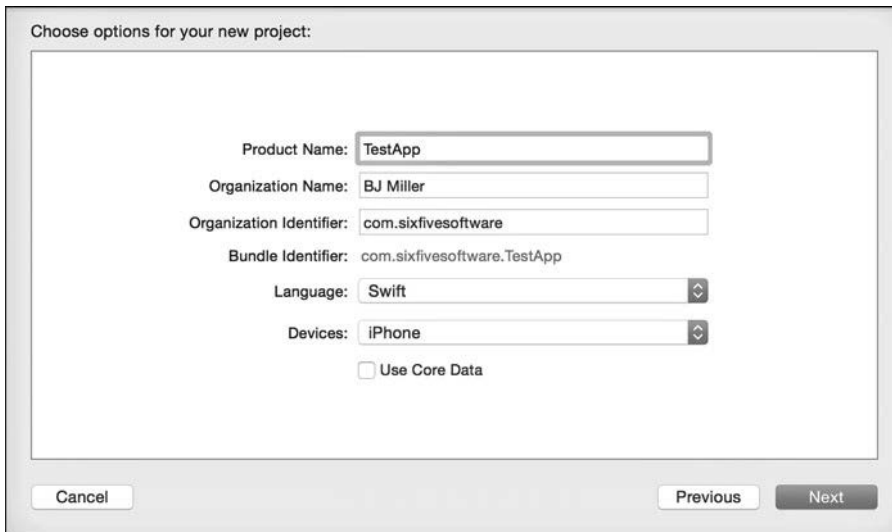
The Welcome to Xcode screen, where you can choose to create or edit projects and playgrounds.

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.



Choose options for your new project:

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

Devices:

Use Core Data

Cancel Previous Next

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.

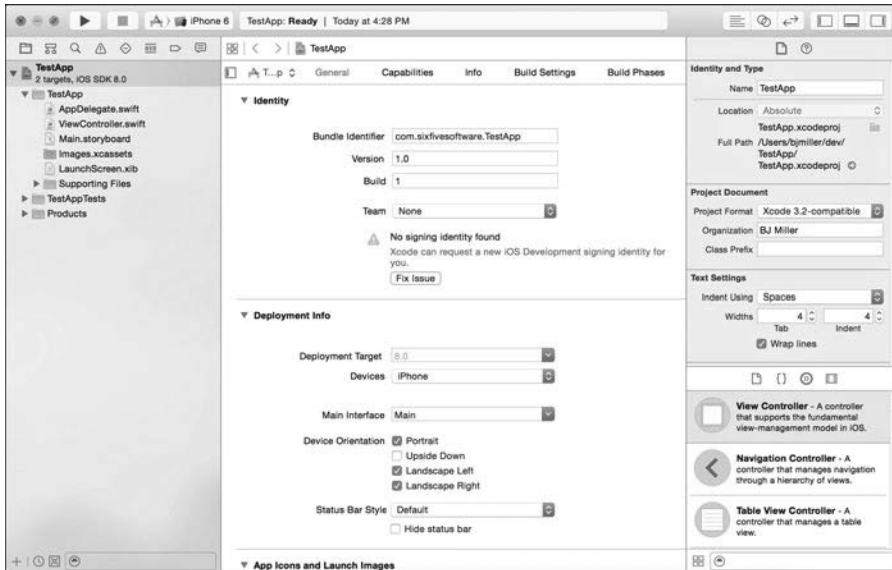


FIGURE 1.4

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 frameworks 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

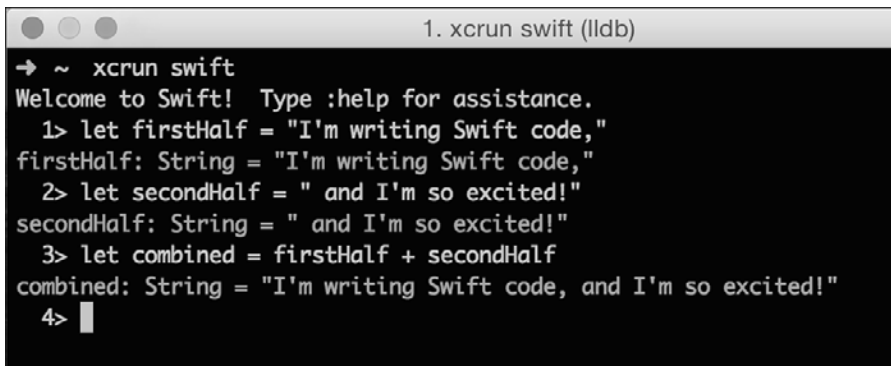
Let's do another example of some Swift code here; hopefully this one isn't too difficult yet. If you don't fully understand it, don't worry; we cover this in great detail in the next hour.

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.

A screenshot of a terminal window titled "1. xcrun swift (lldb)". The terminal shows the following text:

```
→ ~ xcrun swift
Welcome to Swift! Type :help for assistance.
1> let firstHalf = "I'm writing Swift code,"
firstHalf: String = "I'm writing Swift code,"
2> let secondHalf = " and I'm so excited!"
secondHalf: String = " and I'm so excited!"
3> let combined = firstHalf + secondHalf
combined: String = "I'm writing Swift code, and I'm so excited!"
4> |
```

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

- * (asterisks), 331
- { } (curly braces), 101
- ! (exclamation points), removing, 357
- === (identical to operator), 149
- !== (not identical to operator), 149
- 1>, 8
- _ (underscore), 358

A

- access control, type access control, 230-231
- accessing
 - super from subclasses, 163
 - type properties from instance properties, 224
 - values, indexes and subscripts, 41

- actions, 352
- adding
 - constraints to interfaces, 367-369
 - data to dictionaries, 48-49
 - elements to arrays, 42-43
 - elements to interfaces, 355-356
 - populating data, 356-363
 - instance methods
 - to enums, 180-182
 - to structs, 142-143
 - iTunes Search to Songs app, 380-383
 - labels to interfaces, 364-367
 - methods to protocols, 266
 - optional members, to protocols, 274
 - parameters to protocol methods, 267
 - type constraints, 316

adopting protocols, 264-265, 269-273

extensions, 295-296

aliased types, extending, 288

align attribute, 150

allObjects computed property, 325

allocation, Objective-C, 333-335

alternatives, to operation queues, 388

AND operator, 33-34

AnyObject, 238

ARC (Automatic Reference Counting), 247-248

reference behaviors, 248

reference cycles, 248-255

closures, 256

unowned references, 254-255

weak references, 252-253

reference relationships, 248

argument validation, 105

arithmetic operators, 29

Array, 396

Objective-C, 336-337

ArrayLiteralConvertible, 395

arrays, 39-41

accessing values, 41

adding elements, 42-43

data, creating/accessing, 41-42

declaring, 39-40

enumerated arrays, for-in loops, 93

inserting elements, 43

items

removing, 44

updating/replacing, 44

for-in loops, 90-91

for loops, 87

manipulating, 42

methods, 39-46

performing functions on each member, 123

properties, 39-46

slicing, 397

as?, 275

as operator, 275

assigning

constants, 14-15

variables, 14-15

assignment operator, 30

assignments, value types, 139

Assistant Editor, 6

associated types, in protocols, 20-21

associated values

bindings, 178-179

enums, 177

named parameters, 177

asterisks (*), 331

asynchronous execution

paths, 379

asynchronous programming

concurrency, 376-380

operation queues, 379

asynchronous results,

observing, 377

attributes, align attribute, 150

Attributes Inspector, 356-357

auto-complete feature, Xcode, 125

Automatic Reference

Counting. *See* ARC

(Automatic Reference Counting)

B

base classes

inheritance, 156-157

overriding instance

methods, 160-161

behaviors

protocols, 277

references, ARC

(Automatic Reference Counting), 248

binary operators, 29

assignment operator, 30

comparison operators, 31

compound assignment operators, 30-31

logical operators, 33

range operators, 32-33

remainder operator, 29-30

standard arithmetic

operators, 29

bindings, associated values, 178-179

blue text bubble, 383

Bool, 16, 19-20

boolean values, 394

BooleanLiteralConvertible,
395
break keyword, 71
break statement
 transferring control, 94-95
 transferring control
 of execution, switch
 statement, 76-77
bridging Objective-C, 335
 module bridging, 335
 type bridging, 335-337

C

C, for loops, 86-87
cache expiry, 388
caches, 388
call instance methods,
optional chaining, 309-310
calling
 functions, 100
 instance methods, from
 instance methods, 144
casting, 19
cells, reusing, 357
Character, 21
CharacterLiteralConvertible,
395
checking, for protocol
conformance, 275
choosing, queues, 376
class identity, 166-167
class inheritance, 155,
270-272

class instances
 comparing, 149-150
 copying, 147
class keyword, 267
class types, initializing, 193
classes, 137-139
 comparing to structs,
 139-140
 defining properties,
 140-141
 differences, 147-150
 instance methods,
 141-145
 similarities, 145
 creating with dependent
 properties, 249-250
 deciding when to use,
 150-151
 defining with properties
 and methods, 145-146
 initialization delegation,
 195-196
 sizes, 367
closure capture lists, 256
 resolving reference cycles
 in closures, 256-258
 unowned references, 258
closure expressions
 parameters, 127
 return statements, 127
 shorthand argument
 names, 128
 structure, 125-129
 syntax, 125
 type inference, 127

closures, 124
 lazy stored properties, 208
 reference cycles, 256
 trailing closures, 128
Cocoa, 238
 concurrency, 374
 dispatch queues, 375
 operation queues, 375
 type bridging, 335-336
Cocoa Touch, 238
combining
 logical operators, 35
 strings, 8
Comparable, 398-399
comparing
 class instances, 149-150
 classes and structs,
 139-140
 defining properties,
 140-141
 differences, 147-150
 instance methods,
 141-145
 similarities, 145
 equality, 150
 strings, 395
 values, do-while loops,
 84-85
comparison operators, 31
compiler errors, 186
completionHandler
argument, 383
compound assignment
operators, 30-31
computed instance,
extensions, 287-289

computed properties,
209, 212

concurrency, 373-374

adding iTunes Search,
380-383

asynchronous
programming, 376-380

dispatch queues, 374-375

JSON (JavaScript Object
Notation), 383-390

operation queues, 375

queues, choosing, 376

conditional statements,
comparison operators, 31

conditions, 65

if statement, 65-68

switch statement. *See*
switch statement

ternary conditional
operators, 69-70

while loops, 82

conformance

checking for protocol
conformance, 275

protocols, extensions,
295-296

constants, 13-14

assigning, 14-15

constraints, 352

interfaces, adding,
367-369

continue statement,
transferring control in
loops, 93-94

control

transferring in loops

with break statement,
94-95

with continue
statement, 93-94

transferring with
return, 107

convenience initializers, 195

convertible protocols, 395-396

converting between integers
and floating-point
numbers, 19

copying

struct and class
instances, 147

value types, 148

Core Data, 286

countElements, 21

CPUs (single-core processors),
373-374

custom initializers

enums, 189-190

extensions, 292-293

structs, 189

custom values, initializers,
190-191

D

data

adding to dictionaries,
48-49

arrays, creating/accessing,
41-42

populating, adding
elements to interfaces,
356-363

data types, 15-17

Bool, 19-20

Character, 21

Double, 18-19

finding properties and
methods on, 20-21

Int, 17

String, 20-22

type inference, 16

values, initializing, 22-23

deallocation, 243

declaring

arrays, 39-40

stored properties, 206

variables, 15

decrement operators, 28

default parameter values,
functions, 112

default values

initializers, 190-191

initializing value types,
188-192

deinit, 244

deinitialization, 243-247

delegates, 304

delegation

initialization. *See*
initialization delegation

protocols, 275-280

Dependency Inversion
Principle, 276

dependent properties, classes,
creating, 249-250

- description() method, 167
- designated initializers, 195
- designating variables as optional, 54
- dictionaries, 46
 - data, adding, 48-49
 - initializing, 47-48
 - key-value pairs, 46-47
 - for-in loops, 91-92
 - methods, 50
 - properties, 50
 - removing items, 50
 - type inference, 48
- Dictionary, Objective-C, 337
- DictionaryLiteralConvertible, 395
- dispatch queues, 374-375
- dispatch_sync function, 378
- dispatched closures, nesting, 378
- dispatching
 - network calls, 387
 - tasks, 377
- Double, 16, 18-19, 235, 394
- do-while loops, 83-84
 - comparing values, 84-85
 - random number generators, 85-86
- downcasting, 237-238
- downloading
 - starter project, integrating Swift into Objective-C app, 337-339
 - Xcode, 2

E

- elements, 352
 - adding to arrays, 42-43
 - inserting in arrays, 43
 - interfaces, adding. See interfaces, adding elements
- else, 66-67
- else if, 67
- enumerated arrays, for-in loops, 93
- enums, 171-172
 - associated values, 177
 - custom initializers, 189-190
 - initializing, 193
 - instance methods, adding, 180-182
 - raw values, 173-174
 - setting enum values, 174-175
 - shorthand syntax, 175-176
 - structure, 172-173
 - switching values, 178-179
- equality, comparing, 150
- Equatable protocol, 316, 398-399
- ExtendedGraphemeClusterLiteralConvertible, 396
- extending
 - aliased types, 288
 - generic types, 322-323

extensions

- defining, 285-286
- functionality, 287
 - computed instance, 287-289
 - custom initializers, 292-293
 - instance methods, 289-290
 - nested types, 294-295
 - protocol adoption and conformance, 295-296
 - subscripts, 290-291
 - type methods, 289-290
 - type properties, 287-289
- structure, 286
- external parameter names, 111-112
 - initializers, 192-193

F

- failable initializers, 333
- fallthrough statement, transferring control of execution, switch statement, 77
- fetchData() method, 383
- file structure, Objective-C, 330
 - header files, 330-332
 - implementation files, 332-333

filter(), 401

first-class type, 172

Float, 16, 19

floating point, 394

FloatLiteralConvertible, 396

for loops, 86

arrays, 87

for-condition-increment
loops, 86-87

for-in loops, 88

arrays, 90-91

dictionaries, 91-92

enumerated arrays, 93

iterating through
different data types,
88-90

ranges, 89

strings, 89-90

tuples, 92-93

**forced unwrapping with
unwrap operator, optional
variables**, 55-56

for-condition-increment loops,
86-87

for-in loops, 88

arrays, 90-91

dictionaries, 91-92

enumerated arrays, 93

items, 88

iterating through different
data types, 88-90

ranges, 89

strings, 89-90

tuples, 92-93

fromRaw(T), 175

func keyword, 100

funcName, 101

**function parameters, using
functions as**, 122-124

functionality, extensions, 287

computed instance,
287-289

custom initializers,
292-293

instance methods,
289-290

nested types, 294-295

protocol adoption and
conformance, 295-296

subscripts, 290-291

type methods, 289-290

type properties, 287-289

functions, 100, 129-131

argument validation, 105

calling, 100

creating to return mean,
median, mode, 108-109

default parameter values,
112

filter(), 401

higher order functions,
117-118

returning function types,
118-120

map(), 400

median function, rewriting,
129-131

nesting functions within
functions, 120-122

no parameters and no
return types, 102

in-out parameters,
112-113

with parameters, 103-105

parameters, external
parameter names,
111-112

performing, on each
member of arrays, 123

reduce(), 400-401

reduce function, 126-127

return types, 106-110

return values, ignoring,
110

scopes, 104

standard library functions.
See standard library
functions

structure, 101

syntax, 100-101

times, 104

types, 102-103

variadic parameters,
105-106

G

GCD, 376

tasks, dispatching, 378

**generic stack types,
implementing**, 319

generic types

creating, 319-322

extending, 322-323

generics, 313-314

associated types, in
protocols, 20-21

creating generic functions
to return unique arrays,
317-318

creating generic types,
 319-322
 extending, generic types,
 322-323
 placeholder types,
 314-315
 type constraints,
 specifying, 315-319
 type parameters, 314-315
get keyword, 210
getters, property accessors,
209-210
global functions, 401-402
 sort(), 403-404
 sorted(), 403-404
 split(), 402
 startsWith(), 402
 stride(), 403

H
Hashable protocols, 318
header files, Objective-C,
330-332
hiding, parameter names, in
initializers, 192
higher order functions,
117-118
 functions as function
 parameters, 122-124
 nesting functions within
 functions, 120-122
 returning function types,
 118-120

I
identical to operator, 149
if statement, 65-68
if-else statement, 66
ignoring, return values, 110
implementation files,
Objective-C, 332-333
implementing initialization
chaining, 197-199
implicitly unwrapped
optionals, optional
variables, 57
increment operators, 28
indexes, accessing, values, 41
infinite loops, while loops, 83
inheritance, 155-156
 base classes, 156-157
 class identity, 166-167
 overrides, preventing,
 164-166
 overriding inherited
 methods, 159-162
 subclasses, creating,
 157-159
 super, 162-164
inherited initializers, 196
inheriting
 property accessors,
 215-218
 property observers,
 218-220
 protocols, 269-273
init(), 186

initialization, 185
 Objective-C, 333-335
 order of, Objective-C, 333
initialization chaining,
197-199
initialization delegation,
193-194
 classes, 195-196
 structures, 194-195
initialization process,
196-197, 200-201
initializer chaining, 196
initializers, 187
 convenience initializers,
 195
 designated initializers, 195
 failable initializers, 333
 goal of, 186-187
 inherited initializers, 196
 Objective-C, 331
 setting default and custom
 values, 190-191
initializing
 class types, 193
 dictionaries, 47-48
 enums, 193
 properties, in structs, 141
 value types, 187
 external parameter
 names, 192-193
 setting default values,
 188-192
 values, data types, 22-23
inout keyword, 113

in-out parameters, functions, 112-113

inserting, elements, in arrays, 43

instance methods, 141-145

adding

to enums, 180-182

to structs, 142-143

calling from instance methods, 144

extensions, 289-290

instance variables, 206-207

instances, determining an instance's type, 235-237

Instrument, 237

Int, 16-18, 286, 394

finding ranges, 17-18

IntegerLiteralConvertible, 396

integers, 394

integrating Swift into

Objective-C app, 337

creating classes and bridging headers, 339-343

downloading starter project, 337-339

exposing Objective-C classes to Swift, 343

extending Objective-C classes with Swift, 343-344

running apps, 345

updating classes in storyboard, 344-345

interfaces

actions, 352

adding elements, 355-356
populating data, 356-363

adding labels, 364-367

adding UI constraints, 367-369

building, 353-354

constraints, 352

elements, 352

outlets, 352

projects, creating, 355

scenes, 352

segue, 352, 363

preparing for, 366

view controllers, 352

internal access, 230

iOS, playgrounds, 6

is operator, 275

items

for-in loops, 88

removing

from arrays, 44

from dictionaries, 50

updating/replacing, in arrays, 44

iterating

through arrays, for-in loops, 90-91

through dictionaries, for-in loops, 91-92

through different data types, for-in loops, 88-90

iTunes Search, adding to Songs app, 380-383

J-K

JSON (JavaScript Object Notation), 238

parsing, 383-390

key-value pairs, dictionaries, 46-47

keywords

break, 71

class, 267

deinit, 244

func, 100

get, 210

initializing, 186

inout, 113

lazy, 207

mutating, 267

required, 267

self, 143

static, 267

strong, 331

super, 162-164

typealias, 229

where, 75-76

L

labels, adding to interfaces, 364-367

lazy keyword, 207

lazy stored properties, 207-208

closures, 208

let, 40

libraries, standard library, 393-394

local access, caches, 388

logical AND operator, 33-34

logical NOT operator, 28, 35

logical operators, 33

- AND operator, 33-34
- combining, 35
- NOT operator, 35
- OR operator, 34

logical OR operator, 34

loops

- for loops, 86
- transferring control
 - with break statement, 94-95
 - with continue statement, 93-94
- while loops. *See* while loops

M

Mac, playgrounds, 6

manipulating, arrays, 42

map(), 400

mapEachElement, 124

matching values, switch statement, 71-72

mean function, 125

median function, rewriting, 129-131

memberwise initialization, structs, 188-189

methods

- adding to protocols, 266
- arrays, 39-46
- classes, 145-146
- defining in protocols, 266-267
- dictionaries, 50
- fetchData() method, 383
- naming, 358
- optional chaining, 308-310
- parseJson() method, 385
- protocols, 273-274

Microsoft SharePoint Server, optional chaining, 304

MKAnnotation protocol, 273

module bridging, Objective-C, 335

modulo operator, 29-30

multiple protocol inheritance, 270-272

MutableCollectionType, 397

mutating, struct properties, 148-149

mutating functions, 267

mutating keyword, 267

N

named parameters, associated values, 177

naming methods, 358

naming conventions, protocols, 396

nested functions, 121-122

- values versus references, 121

nested method calls, Objective-C, 334

nested types, extensions, 294-295

nesting, dispatched closures, 378

nesting functions, within functions, 120-122

network calls, dispatching, 387

nil, 53-54, 244

nil coalescing operator, 58

NilLiteralConvertible, 396

non-specific types, type casting, 238

not identical to operator (!==), 149

NOT operator, 35

NSArray, Objective-C, 336-337

NSDictionary, Objective-C, 337

NSNumber, Objective-C, 336

NSOperations, 379-380

NSString, 385-386

- Objective-C, 336

NSURLSession, downloading data, 382

numeric types, 394

- Objective-C, 336

O

Objective-C, 329

- allocation, 333-335
- bridging, 335
 - module bridging, 335
 - type bridging, 335-337

- file structure, 330
 - header files, 330-332
 - implementation files, 332-333
- if statement, 66
- initialization, 333-335
- initializers, 331
- nested method calls, 334
- order of initialization, 333
- versus Swift, 22
- synthesizing, 331
- Objective-C app, integrating Swift into, 337**
 - creating classes and bridging headers, 339-343
 - downloading starter project, 337-339
 - exposing Objective-C classes to Swift, 343
 - extending Objective-C classes with Swift, 343-344
 - running apps, 345
 - updating classes in storyboard, 344-345
- Objective-C classes, exposing to Swift, 343**
- Objective-C properties, 206**
- observing**
 - asynchronous results, 377
 - serial results, 377
- Open/Closed Principle, 286**
- operation queues, 375**
 - alternatives, 388
 - asynchronous programming, 379
- operators, 27**
 - binary operators, 29
 - assignment operator, 30
 - comparison operators, 31
 - compound assignment operators, 30-31
 - logical operators, 33
 - modulo operator, 29-30
 - range operators, 32-33
 - remainder operator, 29-30
 - standard arithmetic operators, 29
 - nil coalescing operator, 58
 - ternary conditional operators, 35
 - unary operators, 27
 - increment and decrement operators, 28
 - logical NOT operator, 28
 - unary minus operator, 29
- optional chaining**
 - call instance methods, 309-310
 - defining, 301-302
 - methods, 308-310
 - properties, 302-303
 - subscripts, 303-308
- optional values, 53-54**
 - implicitly unwrapped optionals, 57
 - use cases, 58-61
- variables
 - forced unwrapping with unwrap operator, 55-56
 - nil coalescing operator, 58
 - optional binding to unwrap, 56
 - wrapping/unwrapping, 54-55
- optional variables**
 - forced unwrapping with unwrap operator, 55-56
 - implicitly unwrapped optionals, 57
 - unwrapping, 54-55
 - wrapping, 54-55
- optionals, 23**
- OR operator, 34**
- order of initialization, Objective-C, 333**
- outlets, 352**
- overloading**
 - operators and protocols, 398-399
 - subscripts, 234-235
- overrides**
 - preventing, 164-166
 - property accessors, 215-218
- overriding**
 - inherited methods, 159-162
 - instance methods, 160-161
 - property observers, 217-220
 - subscripts, 235

P**parameters**

- adding to protocol methods, 267
- closure expressions, 127
- functions, 103-105
 - default parameter values, 112
 - external parameter names, 111-112
- as function parameters, 122-124
- in-out parameters, 112-113
- variadic parameters, 105-106
- named parameters, associated values, 177
- names, hiding, in initializers, 192

parentheses, 70**parseJson() method, 385****parsing, JSON (JavaScript Object Notation), 383-390****performing, functions on each member of arrays, 123****performMathAverage, 120****placeholder types, generics, 314-315****playgrounds**

- Mac, 6
- Xcode, 6-7

Point struct

- creating instances of, 141
- creating to return mean, median, mode, 140

populating data, adding elements to interfaces, 356-363**prepareForSegue, updating, 389****preventing, overrides, 164-166****Printable protocol, 325****private access, 230****projects, interfaces, creating, 355****properties, 205**

- arrays, 39-46
- classes, 145-146
- computed properties, 209, 212
- defining, 140-141
- dictionaries, 50
- initializing in structs, 141
- optional chaining, 302-303
- property accessors, 209
 - getters, 209-210
 - inheriting/overriding, 215-218
 - setters, 210
- property observers, 212-214
 - inheriting/overriding, 218-220
- protocols, 265-266, 273-274
- stored properties, 206
 - instance variables, 206-207
 - lazy stored properties, 207-208
- structs, mutating, 148-149
- type properties, 223-225, 266

property accessors, 209

- getters, 209-210
- inheriting, 215-218
- overriding, 215-218
- setters, 210

property observers, 212-214

- inheriting, 218-220
- overriding, 218-220

protocol names as types, 268-269**protocol<>, 395****protocols, 395-397**

- adding optional members, 274
- adopting extensions, 295-296
- adopting and inheriting, 269-273
- associated types, 20-21
- behaviors, 277
- checking for conformance, 275
- Comparable, 398-399
- conformance, extensions, 295-296
- convertible protocols, 395-396
- creating and adopting, 264-265
- defining properties, 263-264
- delegation, 275-280
- Equatable protocol, 398-399
- methods, 273-274
 - adding, 266
 - defining, 266-267

MutableCollectionType, 397
 naming conventions, 396
 overloading operators and, 398-399
 Printable, 325
 properties, 265-266, 273-274
 protocol names as types, 268-269
 Sliceable, 397
 providing table view data
 source methods, 358-359
 public access, 230

Q

queues
 choosing, 376
 dispatch queues, 374-375
 operation queues, 375
 versus threads, 375

R

random number generators, 85-86
 range matching, switch statement, 72-73
 range operators, 32-33
 ranges, for-in loops, 89
 raw values, enums, 173-174
 setting enum values, 174-175

Read-Eval-Print-Loop (REPL), 7-9
 reduce(), 400-401
 reduce function, 126-127
 reference behaviors, ARC (Automatic Reference Counting), 248
 reference cycles
 ARC (Automatic Reference Counting), 248-255
 unowned references, 254-255
 weak references, 252-253
 closures, 256
 creating, 250
 resolving in closures, with closure capture lists, 256-258
 reference relationships, ARC (Automatic Reference Counting), 248
 reference types, 138
 references, versus values in nested functions, 121
 refining switch cases with where keyword, 75-76
 remainder operator, 29-30
 removing
 ! (exclamation points), 357
 items
 from arrays, 44
 from dictionaries, 50
 repeating code, 235
 REPL (Read-Eval-Print-Loop), 7-9, 249
 replacing items, in arrays, 44

required keyword, 267
 resolving reference cycles in
 closures, closure capture lists, 256-258
 return, transferring control, 107
 return someValue, 101
 return statements, closure expressions, 127
 return types, functions, 106-110
 return values, ignoring, 110
 returning function types, higher order functions, 118-120
 ReturnType, 101
 reusing cells, 357
 rewriting median function, 129-131

S

scenes, 352
 scopes, functions, 104
 segue, 352, 363
 preparing for, 366
 self, 143, 189
 serial results, observing, 377
 setters
 property accessors, 210
 providing custom setters for Square structs, 210-211
 SharePoint Object Model, 304
 shorthand argument names, closure expressions, 128

- shorthand syntax, enums, 175-176
- size classes, 367
- Slice**, 396
- Sliceable**, 397
- slicing arrays, 397
- Song struct**, 362
- SongDetailViewController class**, 362
- sort()**, 403-404
- sorted()**, 403-404
- specifying type constraints, 315-319
- split()**, 402
- Square struct**, providing custom setters, 210-211
- Stack**, 320
- standard arithmetic operators, 29
- standard library, 393-394
- standard library functions, 394
 - filter()**, 401
 - global functions, 401-402
 - sort()**, 403-404
 - sorted()**, 403-404
 - split()**, 402
 - startsWith()**, 402
 - stride()**, 403
 - map()**, 400
 - numeric types, 394
 - protocols, 395-397
 - overloading operators and, 398-399
 - reduce()**, 400-401
- Slice**, 396
 - string type, 394
 - string comparison, 395
- startsWith()**, 402
- static keyword**, 267
- stored properties**, 206
 - instance variables, 206-207
 - lazy stored properties, 207-208
- stride()**, 403
- String**, 16, 20-22
 - Objective-C, 336
- string comparison**, 395
- string type**, 394
 - string comparison, 395
- StringLiteralConvertible**, 396
- strings**
 - combining, 8
 - for-in loops, 89-90
- strong keyword**, 331
- struct instances**, copying, 147
- structs**
 - adding instance methods, 142-143
 - comparing to classes, 139-140
 - differences, 147-150
 - instance methods, 141-145
 - similarities, 145
 - custom initializers, 189
 - deciding when to use, 150-151
 - initializing properties, 141
 - memberwise initialization, 188-189
 - overview, 137-139
 - properties, mutating, 148-149
 - type properties, 224
- structures**
 - closure expressions, 125-129
 - comparing to classes, defining properties, 140-141
 - enums, 172-173
 - extensions, 286
 - functions, 101
- initialization delegation**, 194-195
- subclasses**
 - creating, 157-159
 - overriding instance methods, 160-161
- subscripts**, 156, 231-234
 - accessing values, 41
 - creating, 231
 - extensions, 290-291
 - optional chaining, 303-308
 - overloading, 234-235
 - overriding, 235
- super**, 162-164
- superclass method**, preventing overrides, 164-165
- superclasses**, 157
- Swift**, 2
 - versus Objective-C, 22
- Swift REPL**, 102

switch cases, where keyword, 75-76

switch statement, 70-71

matching values, 71-72

range matching, 72-73

refining switch cases with where keyword, 75-76

transferring control of execution, 76

break statement, 76-77

fallthrough statement, 77

tuple matching, 74-75

switching enum values, 178-179

syntax

closure expressions, 125

do-while loops, 83

functions, 100-101

for loops, 86-87

while loops, 82

synthesizing, Objective-C, 331

T

table view controller, 357

table view data source

methods, providing, 358-359

table views, 275

tableView, 386

tasks, dispatching, 377

ternary conditional operators, 35, 69-70

threads, versus queues, 375

times, functions, 104

title property, 265

trailing closures, 128

transferring

control, with return, 107

control in loops

with break statement, 94-95

with continue statement, 93-94

control of switch statements, 76

break statement, 76-77

fallthrough statement, 77

transitions, 352

tuple matching, switch statement, 74-75

tuples, 50-51

for-in loops, 92-93

type access control, 230-231

type aliasing, 229

type bridging, Objective-C, 335-337

type casting, 235-237

downcasting, 237-238

non-specific types, 238

type check operator, 235

type constraints

adding, 316

specifying, 315-319

type inference, 16

closure expressions, 127

dictionaries, 48

type methods, 175, 225-228

extensions, 289-290

type parameters, generics, 314-315

type properties, 223-228, 266

accessing from instance properties, 224

extensions, 287-289

structs, 224

typealias keyword, 229, 288

types

determining an instance's type, 235-237

downcasting, 237-238

functions, 102-103

protocol names as types, 268-269

U

UI (user interface), 351

UI constraints, adding to interfaces, 367-369

UInt, 17

unary minus operator, 29

unary operators, 27

increment and decrement operators, 28

logical NOT operator, 28

unary minus operator, 29

underscore (_), 358

Unicode character, 15

unowned references

closure capture lists, 258

reference cycles, 254-255

unsigned integer, 17

unwrapping

- optional variables, 54-55
- variables, optional binding to unwrap, 56

updating

- items in arrays, 44
- prepareForSegue, 389

use cases, optional values, 58-61**user interface (UI), 351-352.**
See also interfaces**V****value types**

- assignments, 139
- copying, 148
- initializing, 187
 - external parameter names, 192-193
 - setting default values, 188-192

values

- comparing, do-while loops, 84-85
- initializing, data types, 22-23
- in nested functions, versus references, 121

var, 40, 265**variables, 14**

- assigning, 14-15
- declaring, 15
- designating as optional, 54
- optional binding to unwrap, 56
- optional variables, nil coalescing operator, 58

variadic parameters, functions, 105-106**view controllers, 352, 356-357****viewDidLoad() method, 376****W****weak references, reference cycles, 252-253****where keyword, switch cases, 75-76****while loops, 81-83**

- conditions, 82
- do-while loops, 83-84
 - comparing values, 84-85
 - random number generators, 85-86
- infinite loops, 83
- while loops, 82-83

wrapping optional variables, 54-55**X-Y-Z****Xcode, 2**

- auto-complete feature, 125
- downloading, 2
- overriding, 159
- overview, 2-5
- playgrounds, 6-7

Xcode command line tools, 249**Xcode playgrounds, ARC (Automatic Reference Counting), 249****xcrun, 8**