Contents at a Glance

Introduction ........................................................................................................... xiv

HOUR 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 ................................................................. 39
  5 Controlling Program Flow with Conditionals ............................................ 59
  6 Understanding Optional Values ................................................................. 77
  7 Iterating Code with Loops .......................................................................... 89
  8 Using Functions to Perform Actions ............................................................ 107
  9 Understanding Higher Order Functions and Closures ............................... 127
 10 Learning About Structs and Classes .......................................................... 147
 11 Implementing Class Inheritance ............................................................... 165
 12 Harnessing the Power of Enums ................................................................ 183
 13 Customizing Initializers of Classes, Structs, and Enums ............................ 199
 14 Digging Deeper with Properties ................................................................. 221
 15 Adding Advanced Type Functionality ...................................................... 239
 16 Understanding Memory Allocation and References .................................... 261
 17 Using Protocols to Define Behavior .......................................................... 279
 18 Using Extensions to Add Type Functionality ............................................ 301
 19 Working with Optional Chaining ............................................................... 315
 20 Introducing Generics ................................................................................. 327
 21 Understanding Protocol-Oriented Programming ....................................... 343
 22 Handling Errors ......................................................................................... 361
 23 Adding Interoperability with Objective-C ............................................... 377
 24 Functional Thinking in Swift ..................................................................... 403

Index ..................................................................................................................... 421
Table of Contents

Introduction .................................................. xiv
  Who Should Read This Book? ................................. xv
  What Should You Expect from This Book? ................. xvi

HOUR 1: Introducing the Swift Development Environment .............. 1
  What Is Swift? .................................................. 1
  Getting Started ................................................. 2
  Summary ........................................................ 10
  Q&A .............................................................. 11
  Workshop ....................................................... 11
  Exercise ........................................................ 12

HOUR 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

HOUR 3: Using Operators in Swift ................................... 27
  Unary Operators ............................................... 27
  Binary Operators ............................................. 29
  Ternary Conditional Operators ............................. 35
  Summary ........................................................ 36
  Q&A .............................................................. 36
  Workshop ....................................................... 37
  Exercise ........................................................ 37

HOUR 4: Working with Collection Types ............................... 39
  Arrays .......................................................... 39
  Dictionaries .................................................... 47
# Table of Contents

- Sets .................................................. 51
- Tuples ............................................... 55
- Summary ........................................... 56
- Q&A .................................................. 56
- Workshop ......................................... 57
- Exercise ........................................... 57

## HOUR 5: Controlling Program Flow with Conditionals. ........................................ 59
- The if Statement ..................................... 59
- The switch Statement ............................... 64
- Summary ........................................... 73
- Q&A .................................................. 74
- Workshop ......................................... 74
- Exercise ........................................... 75

## HOUR 6: Understanding Optional Values. ......................................................... 77
- What Are Optional Values? ...................... 77
- How to Designate a Variable as Optional. .... 78
- Wrapping and Unwrapping Optional Variables .... 79
- Use Case for Optionals ............................ 85
- Summary ........................................... 87
- Q&A .................................................. 87
- Workshop ......................................... 87
- Exercise ........................................... 88

## HOUR 7: Iterating Code with Loops. ......................................................... 89
- Two Categories of Loops ......................... 89
- Transferring Control in Loops .................... 102
- Summary ........................................... 104
- Q&A .................................................. 104
- Workshop ......................................... 104
- Exercise ........................................... 106

## HOUR 8: Using Functions to Perform Actions ............................................. 107
- The Nature of Functions in Swift ............... 108
- General Function Syntax and Structure ....... 108
Functions with No Parameters and No Return Type .......................... 110
The Type of a Function .................................................. 110
Functions with Parameters ............................................. 111
Functions with Variadic Parameters .................................. 113
Functions with Return Types .......................................... 114
External Parameter Names .............................................. 118
Default Parameter Values .............................................. 120
Change Argument Values with In-Out Parameters .................... 120
Exiting Early ............................................................. 121
Deferring Execution ...................................................... 122
Summary ................................................................. 123
Q&A ........................................................ ............... 124
Workshop ................................................................. 124
Exercise ................................................................. 125

HOUR 9: Understanding Higher Order Functions and Closures .............. 127
Higher Order Functions .................................................. 127
Closures ................................................................. 134
Summary ................................................................. 143
Q&A ........................................................ ............... 144
Workshop ................................................................. 144
Exercise ................................................................. 145

HOUR 10: Learning About Structs and Classes ................................. 147
Overview of Structs and Classes in Swift ................................ 147
What Swift Structs and Classes Have in Common ....................... 149
Differences Between Structs and Classes ............................... 157
When to Use a Class or a Struct ...................................... 160
Summary ................................................................. 161
Q&A ........................................................ ............... 161
Workshop ................................................................. 162
Exercise ................................................................. 163

HOUR 11: Implementing Class Inheritance ..................................... 165
What Is Inheritance? ...................................................... 165
Identifying a Base Class .................................................. 166
Creating a Subclass ...................................................... 167
<table>
<thead>
<tr>
<th>Hour</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Harnessing the Power of Enums</td>
<td>183-221</td>
</tr>
<tr>
<td>13</td>
<td>Customizing Initializers of Classes, Structs, and Enums</td>
<td>199-219</td>
</tr>
<tr>
<td>14</td>
<td>Digging Deeper with Properties</td>
<td>221-228</td>
</tr>
</tbody>
</table>
# Table of Contents

- Inheriting and Overriding Accessors ........................................ 231
- Inheriting and Overriding Observers ........................................ 233
- Summary ............................................................................. 236
- Q&A ................................................................................. 236
- Workshop ............................................................................ 237
- Exercise .............................................................................. 237

**HOUR 15: Adding Advanced Type Functionality** .............................. 239
- Type Properties and Methods .................................................. 239
- Type Aliasing ....................................................................... 246
- Type Access Control .............................................................. 247
- Subscripts .......................................................................... 249
- Type Casting and Non-Specific Types ...................................... 253
- Summary ............................................................................. 257
- Q&A ................................................................................. 258
- Workshop ............................................................................ 258
- Exercise .............................................................................. 259

**HOUR 16: Understanding Memory Allocation and References** .......... 261
- Deinitialization .................................................................... 261
- Automatic Reference Counting .............................................. 265
- Summary ............................................................................. 275
- Q&A ................................................................................. 276
- Workshop ............................................................................ 277
- Exercise .............................................................................. 277

**HOUR 17: Using Protocols to Define Behavior** .............................. 279
- Defining Protocols .................................................................. 279
- Creating and Adopting Protocols ............................................ 280
- Properties ............................................................................ 281
- Defining Methods in Protocols .............................................. 282
- Using Protocol Names as Types .............................................. 284
- Adopting and Inheriting Multiple Protocols .............................. 285
- Optional Protocol Properties and Methods .............................. 290
- How to Check for Protocol Conformance ................................. 291
- Using Protocols for Delegation .............................................. 292
- Summary ............................................................................. 297
<table>
<thead>
<tr>
<th>HOUR 18: Using Extensions to Add Type Functionality</th>
<th>301</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining Extensions</td>
<td>301</td>
</tr>
<tr>
<td>Adding Functionality with Extensions</td>
<td>303</td>
</tr>
<tr>
<td>Summary</td>
<td>313</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>313</td>
</tr>
<tr>
<td>Workshop</td>
<td>313</td>
</tr>
<tr>
<td>Exercise</td>
<td>314</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 19: Working with Optional Chaining</th>
<th>315</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining Optional Chaining</td>
<td>315</td>
</tr>
<tr>
<td>Chaining Optional Properties</td>
<td>316</td>
</tr>
<tr>
<td>Subscripts</td>
<td>318</td>
</tr>
<tr>
<td>Methods</td>
<td>322</td>
</tr>
<tr>
<td>Summary</td>
<td>324</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>325</td>
</tr>
<tr>
<td>Workshop</td>
<td>325</td>
</tr>
<tr>
<td>Exercise</td>
<td>326</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 20: Introducing Generics</th>
<th>327</th>
</tr>
</thead>
<tbody>
<tr>
<td>An Introduction to Generics</td>
<td>327</td>
</tr>
<tr>
<td>Type Parameters and Placeholder Types</td>
<td>328</td>
</tr>
<tr>
<td>Specifying Type Constraints</td>
<td>329</td>
</tr>
<tr>
<td>Creating Generic Types</td>
<td>333</td>
</tr>
<tr>
<td>Extending Generic Types</td>
<td>336</td>
</tr>
<tr>
<td>Using Associated Types in Protocols</td>
<td>337</td>
</tr>
<tr>
<td>Summary</td>
<td>339</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>340</td>
</tr>
<tr>
<td>Workshop</td>
<td>340</td>
</tr>
<tr>
<td>Exercise</td>
<td>341</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 21: Understanding Protocol-Oriented Programming</th>
<th>343</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Quick Recap of Protocols</td>
<td>343</td>
</tr>
<tr>
<td>Implementing Protocols</td>
<td>344</td>
</tr>
<tr>
<td>Introducing Protocol Extensions</td>
<td>344</td>
</tr>
<tr>
<td>Hour</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>22</td>
<td>Handling Errors</td>
</tr>
<tr>
<td>23</td>
<td>Adding Interoperability with Objective-C</td>
</tr>
<tr>
<td>24</td>
<td>Functional Thinking in Swift</td>
</tr>
<tr>
<td></td>
<td>Index</td>
</tr>
</tbody>
</table>
About the Author

**BJ Miller** is an iOS developer for DXY Solutions, a mobile, web, and design 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 encompassed 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 in paired-programming style. BJ often presents iOS-related topics at CocoaHeads and also speaks at other conferences such as MacTech, 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... again. While this book was largely an update versus brand-new content, it was still an exhausting process. 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), Valerie Shipbaugh (Technical Editor), Paula Lowell (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.

You can email or write directly 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 stronger.

Please note that we cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail we receive, we might not be able to reply to every message.

When you write, please be sure to include this book’s title and author, as well as your name and contact information.

Email: errata@informit.com

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

Reader Services

Register your copy of Sams Teach Yourself Swift in 24 Hours at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process go to informit.com/register and log in or create an account.* Enter the product ISBN 9780672337659 and click submit. Once the process is complete, you will find any available bonus content under “Registered Products.”

*Be sure to check the box that you would like to hear from us in order to receive exclusive discounts on future editions of this product.
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 purchased 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 has evolved so much over the past year that it is difficult sometimes to see any similarities. 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, as well as reintroducing itself at WWDC 2015 as a protocol-oriented programming language.

Swift is officially at version 2.0 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 2.0 and Xcode 7. 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/ExamplesV2.

Swift is already proving to be a great language and as of its release is compatible with iOS 7 and up. Swift can also be written for apps running on OS X Yosemite and later. 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). You should write apps via 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 2.0 and Xcode 7, 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/ExamplesV2.
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.
Operators are a key ingredient to every app. In fact, it’s difficult to write an app—other than the standard “Hello, world!” example—that doesn’t contain at least one operator. Operators are symbols or a set of symbols that change or assign values, combine values, or check or verify values in your code. In this hour, we learn about most of the basic operators in Swift and understand why and how to use them.

All operators in Swift can be categorized as unary, binary, or ternary. Unary operators perform an action upon a single operand, binary operators perform an action upon two operands, and ternary operators perform an action involving three operands.

## Unary Operators

A unary operator operates on a single value. A unary operator can be prefix or postfix, meaning that it can come before a variable or constant (prefix, such as `++count`), or immediately follow a variable or constant (postfix, such as `count++`). Some unary operators can be either (prefix or postfix), while some can be only one or the other. A unary operator cannot have any whitespace between itself and the variable or constant. Unary operators act upon numeric and Boolean types in Swift. Let’s take a look at Swift’s unary operators.
Increment and Decrement Operators

Two similar operators are the increment operator and the decrement operator. The increment operator, denoted by `++`, increases a numeric value by 1, and the decrement operator, denoted by `--`, decreases a numeric value by 1. The increment operator is short-hand for a longer expression; `a++` is the same as `a = a + 1`, and `a--` is the same as `a = a - 1`. Both the increment and decrement operators can be prefix or postfix.

There is a key difference in behavior, however, between the increment and decrement operators concerning prefix and postfix. That is the order in which the value is incremented or decremented and when assignment occurs. A prefixed decrement operator decrements the numeric value by 1 and returns the newly assigned value. Likewise, a prefixed increment operator increments the numeric value by 1 and returns the newly assigned value. The postfixed decrement operator first returns the numeric value before decrementing; likewise, the postfixed increment operator returns the numeric value before incrementing. Take a look at the following code example to see this behavior in action:

```swift
var x = 1
print(++x) // ++x - x increments first, new value is displayed
x = 1 // just resetting value for demonstration
print(x++) // now x is incremented, but after the print function has already occurred
print(x) // notice the different value now when this line is printed?
```

Logical NOT Operator

The logical NOT operator inverts the value of a Boolean variable. In other words, if a variable’s value is `true`, the variable will now be `false`, and vice versa. The logical NOT operator is always a prefix operator. We discuss the logical NOT operator and other logical operators a little more in depth later in this hour. Here is what the logical NOT operator looks like in code:

```swift
let a = true // a is a Bool equaling true
let b = !a // b is a Bool equaling false
```

CAUTION

**The Logical NOT Operator Must Be a Prefix Operator**

Using (!) after a special type of variable, called an optional value, has another meaning, which is covered in Hour 6, “Understanding Optional Values.”
Unary Minus Operator

The unary minus operator is short-hand for multiplying a variable’s value by -1, and returning the result. The unary minus operator is always a prefix operator and can prefix variables, constants, or numeric literals. Here is what the unary minus operator looks like in code:

```swift
let c = 5
let d = -c // d is equal to -5
let e = -42 // e is equal to -42
```

Swift also has a unary plus operator, by prefixing a + before any variable or constant, but it doesn’t actually change the value of a numeric variable, because any negative multiplied by a positive is still a negative. The unary plus operator is more for distinction in your code to show that a certain value should be positive. It is very rarely used, but it is available.

Binary Operators

Binary operators are operators that affect operands, which are the values on either side of the operator. For example, take a + b. In this expression, a and b are operands, and + is the operator. All Binary operators are infix, meaning they are in between two operands.

Standard Arithmetic Operators

Swift provides the four standard arithmetic operators that are binary operators for all numeric types, which are addition (+), subtraction (-), division (/), and multiplication (*). To use these operators in code, you use them exactly the way you would write them in a math equation:

```swift
let a = 5 + 5 // a equals 10
let b = 3 - 2 // b equals 1
let c = 8.0 / 5.0 // c equals 1.6
let d = 4 * 4 // d equals 16
```

Remainder Operator

Swift includes a remainder operator, annotated by the % sign. In other languages, this operator is called the modulo operator, which has slightly different behavior. Swift calculates the remainder operator by figuring out how many times b evenly divides a, and returns the remainder. In other words, a = (b * multiplier) + remainder, with multiplier being the largest number b could multiply against to not produce a value higher than a.

In languages like C and Objective-C, the modulo operator behaved similarly, but unlike Swift, it could not perform against floating-point numbers; you would get a compiler error as the modulo operator cannot take floating-point numbers as its operands. This is another
reason that Swift’s remainder operator is truly a remainder operator, because Swift gives the remainder of what is left over from \(a - (b \times \text{multiplier})\). Listing 3.1 displays the modulo operator in C.

**LISTING 3.1  Modulo Operator in C and Objective-C**

```c
int a = 8;
int b = 6;
int c = a % b; // returns 2, as expected
int d = -8;
int e = d % b; // returns -2, as expected
```

In C and Objective-C, the modulo operator is pretty straightforward, as long as you’re providing integers. Listing 3.2 displays the remainder operator in Swift, which not only works with integers like C, but can also operate on Double types.

**LISTING 3.2  Remainder Operator in Swift**

```swift
let a = 8
let b = 6
let c = a % b // returns 2, just as in C
let d = 8.0
let e = 2.5
let f = d % e // returns 0.5, Swift allows remainders for Double types
```

**The Assignment Operator**

There is just one assignment operator in Swift, as in other languages, and that is the equal sign (=). The assignment operator takes the calculated or literal value on the right side of the equal sign and assigns it to the variable or constant on the left side of the equal sign. We have seen the assignment operator many times throughout this book already, and every single line in Listing 3.1 and Listing 3.2 utilizes the assignment operator.

**Compound Assignment Operators**

Compound assignment operators in Swift utilize the previously mentioned assignment operator (=), coupled with another operator, to perform a combined effect. Table 3.1 details each operator, what code it is short for, and a description.
### Binary Operators

**TABLE 3.1** Compound Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Longer Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>a += b</td>
<td>a = a + b</td>
<td>Add, then assign</td>
</tr>
<tr>
<td>-=</td>
<td>a -= b</td>
<td>a = a - b</td>
<td>Subtract, then assign</td>
</tr>
<tr>
<td>*=</td>
<td>a *= b</td>
<td>a = a * b</td>
<td>Multiply, then assign</td>
</tr>
<tr>
<td>/=</td>
<td>a /= b</td>
<td>a = a / b</td>
<td>Divide, then assign</td>
</tr>
<tr>
<td>%=</td>
<td>a %= b</td>
<td>a = a % b</td>
<td>Compute remainder, then assign</td>
</tr>
</tbody>
</table>

Table 3.1 shows some of the basic compound operators. However, there are many more compound operators, all of which are more advanced than the scope of this book, such as bitwise and logical compound operations. For more information on these operators, see the chapter on expressions in *The Swift Programming Language*, Apple, 2015.

### Comparison Operators

Comparison operators in Swift are similar to those of other programming languages. Each one returns a Boolean value describing whether the expression was true or false. Table 3.2 lists the basic comparison operators.

**TABLE 3.2** Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>a &gt; b</td>
<td>Returns true if a is greater than b, else false.</td>
</tr>
<tr>
<td>&lt;</td>
<td>a &lt; b</td>
<td>Returns true if a is less than b, else false.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>a &gt;= b</td>
<td>Returns true if a is greater than or equal to b, else false.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>a &lt;= b</td>
<td>Returns true if a is less than or equal to b, else false.</td>
</tr>
<tr>
<td>==</td>
<td>a == b</td>
<td>Returns true if a is equal to b, else false.</td>
</tr>
<tr>
<td>!=</td>
<td>a != b</td>
<td>Returns true if a is not equal to b, else false.</td>
</tr>
</tbody>
</table>

As with comparison operators, there are other advanced operators. We cannot discuss them all in this book, but we cover two more of them in Hour 10, “Learning About Structs and Classes.”
**NOTE**

**Using Comparison Operators**

Most often, comparison operators are used in conditional statements, such as an if block, or in loop statements, such as a while block. Examples would be if `a > b { print("a is greater than b") }`, or while `a < 100 { ++a }`. We cover conditionals and control flow in Hour 5, “Controlling Program Flow with Conditionals,” and loops in Hour 7, “Iterating Code with Loops.”

---

**Range Operators**

Range operators in Swift are denoted by two dots and a left angle bracket (..<) or three dots (...), depending on what type of range you need. There are two types of range operators: half-closed range operators and closed range operators.

A half-closed range operator uses two dots and a left angle bracket to indicate a range of integers from its left-hand operand up to but not including the right-hand operand. The expression `1..<5` defines a range of four integers, 1 through 4, but not including 5. You can also use variables or constants with range operators, such as `1..<a`. The half-closed operator is called as such because its range contains its initial value but not the final value.

A closed range operator uses three dots to indicate a range of integers from its left-hand operand up to and including the right-hand operand. The expression `-2...2` defines a range of five integers, from -2 to +2. The closed range operator can also use variables or constants as operands, such as `a...b`. The closed range operator is called closed range because the range is closed on each end by including both operands.

Both half-closed range operators and closed range operators are useful in iterative expressions. Closed range operators are good for ranges where you want both the first and last values of the range to be included. Half-closed range operators are especially good for zero-based lists such as arrays, where the index of the first element is 0, followed by 1, then 2, and so on. Arrays are discussed in more detail in Hour 4, “Working with Collection Types.” Consider the code in Listing 3.3—it illustrates the differences in the two range operators.

**LISTING 3.3 Different Range Operators**

```swift
for a in 1...5 {
    print(a)
}
// This closed range prints the following:
// 1
// 2
// 3
// 4
// 5
```
let arrayLength = 3
for b in 0..<arrayLength {
    print(b)
}
//This half-closed range prints the following:
//0
//1
//2

CAUTION

Ranges Must Progress in a Positive Direction
Ranges in Swift must progress in a positive direction. The starting index (left-hand operand) must be less than or equal to the ending index (right-hand operand). Attempting to progress in a negative direction will result in a compiler error if detected when coding or a run-time error and crash if detected while the app is running. You might want to validate that the left-hand operand is less than the right-hand operand if your operands are not known at compile time.

Logical Operators
Swift provides three logical operators, AND (&&, OR(||), and NOT (!). These operators are called logical operators because they evaluate upon and return Boolean value types based on logic. The three logical operators compare the bits inside their given operands, perform the operation against them, and then return the result. These three operators are similar to those you would find in other languages, but we discuss them here if you’re not familiar.

Logical AND Operator
The logical AND operator, sometimes referred to as just AND, performs a logical AND operation against both operands and returns the result. Let’s illustrate how AND operates and comes up with a return value. Look at the truth table in Table 3.3.

TABLE 3.3 Logical AND Operator Truth Table

<table>
<thead>
<tr>
<th>Operand 1</th>
<th>Operand 2</th>
<th>AND Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Let’s say you’re comparing two Boolean values, a and b, in the following code sample:

```swift
let a = true
let b = false
let c = a && b
```

What value does c have? You’re right; c is false. The AND operator is useful when every value in a certain set of conditions needs to be met first before executing an expression or set of expressions. For example, the following code snippet illustrates what use of the AND operator looks like if you were to try to get access to a device’s motion data:

```swift
if CMPedometer.isStepCountingAvailable() && userAcceptedUseOfMotionProcessor {
    /* it's now ok to get step data */
} else {
    print("cannot get step data")
}
```

If step counting is not available (known by the Boolean return value of CMPedometer.

```swift
isStepCountingAvailable()`) or if the user has explicitly stated it’s not okay for us to have access to the device’s motion coprocessor, then we cannot count the user’s steps.

### Logical OR Operator

The logical OR operator is similar in syntax to the AND operator but behaves slightly differently. The OR operator evaluates two operands and decides whether either one is true or not, no matter which one, and then returns its result. The truth table in Table 3.4 details the OR operator’s behavior.

<table>
<thead>
<tr>
<th>Operand 1</th>
<th>Operand 2</th>
<th>OR Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

The OR operator returns true if either operand in the expression is true. Let’s take another look at a simple code snippet to illustrate OR:

```swift
let a = true
let b = false
let c = a || b
```
What is the value of \( c \)? Again, you’re right; the value is true. A more realistic example might look something like this:

```swift
if userHasCellularConnectivity || userHasWifiConnectivity {
    // assume user has an internet connection, start talking to web service
} else {
    // warn user to enable connectivity hardware
}
```

In the previous example, if either of the two operands (userHasCellularConnectivity or userHasWifiConnectivity) is true, then the first block of code executes. If neither operand is true, the second block of code executes. We cover conditionals with if statements in Hour 5.

**Logical NOT Operator**

The last logical operator, NOT, as mentioned earlier, simply inverts the value of a Boolean variable or constant. Although not a binary operator, NOT is listed here in the logical operators section for completeness. For example, see Table 3.5 for the truth table illustrating the values resulting from the NOT operator.

<table>
<thead>
<tr>
<th>Operand</th>
<th>NOT Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

**Combining Logical Operators**

Logical operators can be combined into larger expressions where multiple lines of logic may be required. The expression \( a \land b \quad \text{||} \quad !c \) combines all three logical operators and can be read as “\( a \) and \( b \) must be true, or not \( c \)”.

While not required, you can insert parentheses to explicitly state the order of operations you want in your code, such as \( (a \land b) \quad \text{||} \quad (!c \land d) \).

**Ternary Conditional Operators**

Swift has only one ternary conditional operator, and it is written as short-hand for a longer expression that checks a particular case or condition and provides a value for the true branch and another value for the false branch. We cover the ternary conditional operator in greater detail in Hour 5, since it is a more succinct substitution for a longer conditional expression.
Summary
In this hour, we covered a lot of material regarding operators. Most operators are straightforward, but if you don’t understand any, don’t worry; operators are used frequently throughout the book, and you will pick it up by repetition. Logical operators manipulate and/or combine Boolean values true and false. Unary operators act upon a single target, binary operators act upon two targets, and ternary operators compare a Boolean value to execute one of two expressions. Operators can be prefix, postfix, or infix, meaning they can be written immediately in front of an operand, immediately after an operand, or in between two operands, respectively. Most of these types of operators are familiar from other languages.

A set of operators in Swift that are new to some C and Objective-C programmers are range operators. These prove to be powerful for iterating over ranges of values, items in a list, and others. Range operators come in two types: half-closed range operators, which include the first but not the last values given, and closed range operators, which include the first and the last values given. We use these operators multiple times throughout this book.

In the next hour, we discuss collection types including Arrays and Dictionaries. Arrays and Dictionaries are crucial to many apps, in that they keep lists of multiple objects in memory to be used for things such as data sources, data retrieved from networks, or even to store preferences between app launches.

Q&A
Q. Is it better to use postfix or prefix?
A. This depends on the needs of your calculations. There is no performance gain using one or the other; it purely depends on how you need to use them. For instance, ++a and a++ both increment a, but the former increments before returning its value, and the latter returns its value and then increments. Be careful of this distinction in your code.

Q. Why are there two range operators? Wouldn’t it be simpler to just use one?
A. Having two range operators allows you to express intent a lot more clearly in code. In the case of a zero-based list, you may have 10 items but the last item’s index is 9, so a half-closed range operator (0..<count, for instance) would be a better fit. If you know you have a finite number of times something should loop, a closed-range operator (1...10, for instance) clearly indicates a range of 1 to 10.

Q. What is a practical example of using remainder?
A. Imagine an app that shows a progress bar when loading. In this case, the progress is only checked occasionally (every nth time) throughout the loading loop, or when the count of the loop % n is equal to 0. This becomes clearer in later hours when we cover conditionals and control flow in Hour 5 and loops in Hour 7.
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 is the difference between postfix and prefix unary operators?
2. In the following code, what is the value of the constant \(c\)?
   ```
   let a = 11
   let b = 5
   let c = a % b
   ```
3. In the following code, what is the value of the variable \(a\)?
   ```
   var a = 5
   let b = 6
   a *= b
   ```
4. In the following code, what is the value of the constant \(b\)?
   ```
   let a = 5
   let b = -a
   ```
5. What is the difference between two dots with a left angle bracket (\(..<\)) and three dots (\(...\)) in a range operation?

Answers
1. The postfix increment does not increase the value of its operand until after it has been evaluated. The prefix decrement increases the value of its operand before it has been evaluated.
2. The value of \(c\) after the remainder operator is 1.
3. The value of \(a\) after the compound assignment is 30.
4. The value of \(b\) after the unary minus operation is -5.
5. The two dot with left angle bracket range (\(..<\)) is a half-closed range because the upper limit is noninclusive. The three dot range (\(...\)) is a closed range because the upper limit is inclusive.

Exercise
Create a new playground. Create two variables to hold two separate numbers, 1 and 10. Use different comparison operators to compare the two values. Next, display the numbers using both the half-closed and closed ranges.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>greater than operator</td>
<td>31</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to operator</td>
<td>31</td>
</tr>
<tr>
<td>++</td>
<td>increment operator</td>
<td>28</td>
</tr>
<tr>
<td>!=</td>
<td>inequality comparison operator</td>
<td>31</td>
</tr>
<tr>
<td>==</td>
<td>is-equal operator</td>
<td>415</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than operator</td>
<td>31</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to operator</td>
<td>31</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical AND operator</td>
<td>33-34</td>
</tr>
<tr>
<td>!</td>
<td>logical NOT operator</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>minus sign</td>
<td>379</td>
</tr>
<tr>
<td>*</td>
<td>multiplication operator</td>
<td>29</td>
</tr>
<tr>
<td>%</td>
<td>remainder operator</td>
<td>29-30</td>
</tr>
<tr>
<td>-&gt; ReturnType</td>
<td></td>
<td>109-110</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
<td>22</td>
</tr>
<tr>
<td>()</td>
<td>parentheses</td>
<td>64</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
<td>379</td>
</tr>
<tr>
<td>?</td>
<td>question mark</td>
<td>78, 208</td>
</tr>
<tr>
<td>%</td>
<td>remainder operator</td>
<td>29-30</td>
</tr>
<tr>
<td>-&gt; ReturnType</td>
<td></td>
<td>109-110</td>
</tr>
<tr>
<td>;</td>
<td>semicolon</td>
<td>22</td>
</tr>
<tr>
<td>-</td>
<td>subtraction operator</td>
<td>29</td>
</tr>
<tr>
<td>-</td>
<td>unary minus operator</td>
<td>29</td>
</tr>
</tbody>
</table>
+ (unary plus) operator, 29
_ (underscore), 68, 206, 383
! (unwrap) operator, 80-81
_ (wildcard underscore), 101

A
Abrahams, Dave, 343
access control
in extensions, 313
internal access, 248
private access, 248
public access, 247-248
types, 247-248, 258
accessing
arrays, 41-42
methods
instance methods on enums, 192
with optional chaining, 322-324
REPL (Read-Eval-Print-Loop), 8
super, 172-174
type properties, 239-241
accessors
inheriting, 231-233
property accessors
getters, 225-226
inheriting, 231-233
overriding, 231-233
overview, 225
setters, 226-228
addition-assignment compound operator, 43
addition (+) operator, 29
adoption of protocols
definition of, 279
multiple protocols, 285-290
syntax, 280-281
aliasing (type), 246-247
allocation, Objective-C, 383-384
AND operator, 33-34
annotation, 16
Any keyword, 256-257
AnyObject keyword, 256-257
appending array elements, 43
append() method, 43
Apple documentation, 417
App Store, downloading Xcode from, 2
closure() function, 66
ARC (Automatic Reference Counting), 262
closures, 272-275
how it works, 265
reference cycles
identifying, 266-269
resolving, 269-272
reference relationships, 265-266
strong references, 265
unowned references, 270-272, 276
weak references, 269-270
area() method, 175
areIntsEqual() function, 329
areStringsEqual() function, 329
areTheseEqual() function, 329-330
arguments, 114
validation, 112
values, changing, 120-121
arithmetic operators, 29
arrays, 332
accessing values in, 41-42
appending elements to, 43
compared to dictionaries, 56
creating, 41-42
declaring, 39-41
for-in loops, 98-99
for loops, 96
functions as members of arrays, 133
generic function to return an array, 331
indexes, 41-42
inserting elements into, 43-44
member types, 40
methods, 46
properties, 46
removing items from, 45-46
replacing items in, 44-45
subscripts, 41-42
tuples, 55-56
type bridging in Objective-C, 386
updating items in, 44-45
zero-based arrays, 41
as keyword, 255-256
as! keyword, 291
as? operator, 291
assignment operator, 14, 30, 44
assignment operator (=), 246
Assistant Editor, 6
associated types, 337-339
associated values
  bindings, 191
  enums, 189-190
  named parameters, 190
asterisk (*), 22, 379
AudioCD struct, 284-285
Automatic Reference Counting.
  See ARC (Automatic Reference Counting)

B
  base classes, 166, 170
  binary operators
    assignment operator, 14, 30, 44
    combinations of logical operators, 35
    comparison operators, 31-32
    compound assignment operators, 30-31
    explained, 29
    logical AND operator, 33-34
    logical NOT operator, 35
    logical OR operator, 34-35
    range operators, 32-33, 36
    remainder operator, 29-30
    standard arithmetic operators, 29
  bindings, associated values, 191
  blocks, do-catch, 367-369
  BluePotion struct, 294
  Book class, 287
  Bool data type, 20
  Boolean types, 20
  Bool type (Objective-C), 399
break keyword, 65, 71, 103
bridging
  definition of term, 384
  module bridging, 384-385
  type bridging, 385-387
    Array and NSArray, 386
    Dictionary and NSDictionary, 387
    Numeric Types and NSNumber, 386
    String and NSString, 386
  bridging header, 384
  Bundle Identifier, 4
  ButtonDelegate class, 293
  ButtonDelegateProtocol, 293
  buttonTapped() method, 293
C
  calling
    functions, 108
    methods, 322-324
capital letters as generic placeholder types, 136
capture, closure capture lists, 272-275
casting down. See unwrapping optionals
catching errors, 366-371
  from throwing function, 367-369
  trying without catching, 369-371
chaining
  chaining functions together explained, 410-411
  flatMap functions, 412-413
  initializer chaining, 211-215
  optional chaining
    definition of term, 315-316
    importance of, 325
    methods, 322-325
    properties, 316-318
    subscripts, 318-322
    use cases, 316
    when to use, 325
checking for protocol conformance, 291
child classes, 167
Circle class, 156
classes, 148. See also inheritance; methods
  base classes, 166, 170
  Book, 287
  ButtonDelegate, 293
  child classes, 167
  Circle class, 156
  class identity, 176-177
  comparing to structs, 149-150, 155-156
  class instance references, 159-160
  differences, 157-158
  instance equality, 160
  instance methods, 151-154
  mutating properties, 158
  properties, 150-151
  creating, 389-393
deinitializing, 261-265, 276
DetailVC, 392
Document, 320
DVD, 287
classes

Employee, 271-272
GameManager, 296
initialization, 207-209, 262-263
default initializer parameters, 207-208
failable initializers, 208-209
initialization delegation, 211-212
Library, 320
MathFunction, 273
Media, 286
NSArray, 79, 386
NSDictionary, 47, 79, 387
NSError, 364
NSMutatableDictionary, 47
NSNumber, 386
NSString, 386
Objective-C classes
exposing to Swift, 394
extending with Swift, 394-395
parent classes, 167
Person, 267
Phone, 266, 269-270
properties, 155-156
Rectangle, 214
Site, 319
SomeClassWithExpensiveOperation, 223
SomeClassWithLazyVar, 223
Square, 215, 232
StackOfPapers, 250, 251
subclasses, 167
creating, 167-168
overriding instance methods, 170-171
superclasses, 167
overriding inherited methods, 169
syntax, 155
TestClass, 263
TheLegendOfZelda, 295
updating in storyboard, 395-396
when to use, 160-161
class keyword, 239
class subscripts, 166
classes
else
else-if syntax, 61-63
if-else syntax, 60-63
where, 87
closed for modification, 301
closed-range operator, 32, 36
closing REPL (Read-Eval-Print-Loop), 9
closure capture lists, 272-275
closures, 134
as parameters, 137
shorthand argument names, 138
single-expression closures, 137
strong reference cycles, 272-275
structure of, 134-137
trailing closures, 139, 142-143
Cocoa error handling, 362-363
collections
arrays
accessing values in, 41-42
appending elements to, 43
compared to dictionaries, 56
creating, 41-42
declaring, 39-41
indexes, 41-42
inserting elements into, 43-44
member types, 40
methods, 46
properties, 46
removing items from, 45-46
replacing items in, 44-45
subscripts, 41-42
tuples, 55-56
updating items in, 44-45
zero-based arrays, 41
definition of term, 39
dictionaries
adding data to, 49-50
compared to arrays, 56
explained, 47
initializing, 48
key-value pairs, 47-48
methods, 50
properties, 50
removing items from, 50
tuples, 55-56
type inference, 48-49
heterogeneous collections, 353-355
homogeneous collections, 353-355
sets
adding elements to, 51-52
conditions
for-condition-increment loop, 94
while loops, 90
configureView() method, 393, 395
conformance (protocol)
checking for, 291
definition of term, 279
constraints, type, 329-332
continue statements, 102-103
controlling program flow.
See program flow, controlling with conditionals
control of execution, transferring
break statement, 71
fallthrough statement, 71-72
convenience initializers, 211
convenience keyword, 211, 308
converting
between integers and floating-point numbers, 19
protocol sequences, 355-357
copying value types, 149
Core Data, 302
counts, reference count, 262
count property, 46
Create a New Xcode Project command, 2
computed properties, 186,
224-225
computed instance properties, 303-305
computed type properties, 241-242
concatenating strings, 21
concatenation (+) operator, 9
corre...
creating. See also declaring; initialization
arrays, 41-42
classes, 389-393
constants, 7, 79
generic types, 333-336
immutable variables, 7
playgrounds, 6
protocol extensions, 346-349
protocols, 280-281
queues, 334-336
reference cycles, 266-269
sets, 51
subscripts, 249-252
variables, 79
Xcode projects, 2-6
curly braces ({}), 109, 279, 302
custom initializers
adding with extensions, 308-309
enums, 203-204
structs, 203
customizing error types, 364-365
custom operators, 416-419

D
data source delegates, 292
data types. See types
deallocation, 261
declaring. See also creating; initialization
arrays, 39-41
constants, 14-15, 22-23
enums, 185
functions, 111
methods
in protocols, 282-284
type methods, 242-246
mutating functions, 283
stored properties, 221-222
variables, 14-15, 22-23
declaration operator (‐), 28
default initializer parameters, 207-208
default initializers, 201
default protocol implementations, overriding, 349-352
default values
default parameter values, 120
default property values, 200
setting, 202-206
defer keyword, 122
deferring execution, 122-123, 371-374
defer statement, 122, 371-374
defining. See declaring
definition of Swift, 1-2
deinitializers, 276
deinitializing classes, 261-265, 276
deinit keyword, 262
delegates
data source delegates, 292
delegate execution, deferring, 372-373
protocols and, 292-297
delegation
initialization delegation
classes, 211-212
definition of term, 209
structures, 210-211
protocols, 292-297
deleting projects, 8
Dependency Inversion Principle, 292
dequeue function, 335
dequeueing, 334-335
description() method, 177
designated initializers, 211, 218
DetailVC class, 392
dictionaries
adding data to, 49-50
compared to arrays, 56
explained, 47
for-in loops, 100
initializing, 48
key-value pairs, 47-48
methods, 50
properties, 50
removing items from, 50
sorting, 100
tuples, 55-56
type inference, 48-49
Dictionary type, 332, 387
didSet keyword, 228, 233, 236
division (/) operator, 29
do-catch blocks, 367-369
Document class, 320
Double data type, 18-19
double forward slash (/\), 7
do-while loops, 90-94
downcasting, 255-256
external parameter names
F

factory methods, 242
failable initializers, 208-209
fall through, 65
fallthrough statement, 71-72
FIFO (First-In-First-Out), 334
files
viewing, 6
Objective-C file structure explained, 378
header files, 378-380
implementation files, 380-382
filter function, 409-410
filtering values
filter function, 409-410
for-in loops, 101-102
final keyword, 174, 231
finding
minimum/maximum ranges of integers, 17
properties/methods of data types, 20-21
first-class types, 184
First-In-First-Out (FIFO), 334
first property, 46
flatMap function, 411-414
chaining together, 412-413
on optionals, 412-414
on sequences, 411
flattening objects
flatMap on optionals, 412-414
flatMap on sequences, 411
floating-point numbers, converting, 19
deferring execution, 122-123
forced unwrapping, 80
force try statement (try!), 369-371, 375
forEach function, 143, 406-409, 419
for loops, 94
arrays, 96
C, 95
for-condition-increment loops, 94-95
for-in loops, 96-101
arrays, 98-99
data types, iterating, 97-100
dictionaries, 100
filtering, 101-102
items, 96
pattern matching, 101-102
ranges, 97
strings, 97-98
tuples, 101
Swift, 95
funcName, 109
functional programming, 127, 403-404. See also functions
defining, 107. See also methods
arc4random(), 66
areIntsEqual(), 329
areStringsEqual(), 329
areTheseEqual(), 329-330
calling, 108
chaining functions together, 410-411
declaring, 111
default parameter values, 120
folders, ObjCInteropTests, 399
forced try statement (try!), 369-371, 375
enqueue, 335
exiting early, 121-122
external parameter names, 118-120
filter, 409-410
flatMap, 411-414
chaining together, 412-413
on optionals, 412-414
on sequences, 411
forEach, 143, 406-409, 419
generic functions
comparing equality with, 329
generic function to return an array, 331
structure of, 328
higher order functions, 127
closures, 134
function parameters, 132-134
nested functions, 130-132
returning, 128-130
in-out parameters, 120-121
instance methods, 135
internal parameter names, 119
length of, 108
map, 404-408, 419
median function, rewriting, 139-140
mutating functions, 283
operator functions, 139
overriding, 357
parameters, 111-112, 132-134
performing as members of arrays, 133
print(), 329
printWeather(), 328
reduce function, 136
return types and functions with no parameters and no return type, 110
functions with return types, 114-118
returning mean, median, mode, 116-117
static dispatch, 349
structure of, 108
styMap, 345
syntax, 109
throwing functions, 367-369
types, 110-111
unique, 331
variadic parameters, 113-114
generic types
creating, 333-336
extending, 336-337
mixing with non-generic parameters, 340
multiple generic types, 340
queues, 334-337
type constraints, 329-332
type parameters, 328-329
placeholder types, 328-329
get keyword, 225-226, 281
getters, 225-226, 379
greater than (>) operator, 31
greater than or equal to (>=) operator, 31
guard let syntax, 371
guard statement, 121-122, 371
H
half-closed range operators, 32, 36
handling errors. See error handling
hashable, 47
Hashable protocol, 331-332
header files (Objective-C), 378-380
Healable protocol, 294
Heart struct, 294
heterogeneous collections, 353-355
hiding parameter names in initializers, 206-207
higher order functions, 127
closures, 134
function parameters, 132-134
nested functions, 130-132
returning, 128-130
homogeneous collections, 353-355
identical to operator, 159
identifying reference cycles, 266-269
id type, 79, 256
if statement, 59-64
C/Objective C versus Swift, 60
else-if syntax, 61-63
if-else syntax, 60-63
if let statement, 80-81
ternary conditional operators, 63-64
ignoring return values, 118
immutable variables, 7, 13
@implementation keyword, 381
implementation files (Objective-C), 380-382
implicitly unwrapped options, 81-82
implicit returns, 137
import Cocoa statement, 6
import UIKit statement, 6
increment operator (++), 28
indexes, 41-42
inequality (!=) comparison operator, 31
inference, type inference, 15-16
inferred type bridging, 385
infinite loops, 91
GameManager class, 296
GameManagerDelegate protocol, 295
generics
associated types, 337-339
explained, 327-328
functions
comparing equality with, 329
generic function to return an array, 331
structure of, 328
infix operators, 29

inheritance
accessing, 172-174
accessors, 231-233
base classes, 166
class identity, 176-177
defined, 165-166
initializers, 211, 218
multiple protocols, 285-290
overriding, 169-172
preventing overrides, 174-176
property accessors, 231-233
property observers, 233-235
protocols, 298
single inheritance, 172
subclasses, 167-168
when to use, 177-178

initialization
class types, 207-209,
262-263
default initializer
class parameters, 207-208
fallable initializers,
208-209
definition of term, 199
dictionaries, 48
initialization delegation
classes, 211-212
definition of term, 209
structures, 210-211
initialization process,
212-217
initializers, 22-23
adding with extensions,
308-309
convenience initializers,
211
default initializers,
201, 207-208
designated initializers,
211, 218
goal of, 200-201
hiding parameter names
in, 206-207
inheritance, 218
inherited initializers, 211
initializer chaining,
211-215
Objective-C, 383-384
properties in structs, 151
safety, 212
value types, 22-23, 41, 201
external parameter
names in initializers,
206-207
setting default values,
202-206
initializers, 22-23
adding with extensions,
308-309
convenience initializers, 211
default initializers, 201,
207-208
designated initializers,
211, 218
goal of, 200-201
hiding parameter names
in, 206-207
inheritance, 218
inherited initializers, 211
initializer chaining, 211-215
init keyword, 200, 208, 383
init() method, 48, 81, 210. See
also initialization
in keyword, 135
in-out parameters, 120-121
input, validating, 85-86
inserting
array elements, 43-44
set elements, 51-52
insert() method, 44
instance methods, 135
adding
to enums, 192-194
with extensions, 305-306
to structs, 152-153
comparing, 151-154
overriding, 170-171
instance properties
accessing type properties
from, 240-241
compared to type properties,
258
instances
deinitializing, 261-265, 276
initializing, 262-263
instancetype keyword, 380
instance variables, 222
Int data type. See integers
integers, 17-18
converting between
integers and floating-point
numbers, 19
extending to adopt and
conform to a protocol,
311-312
finding minimum/maximum
ranges of, 17
unsigned integers, 17
when to use, 18
integrating Swift into Objective-C
creating Swift class and bridg-
ing header, 389-393
downloading starter project,
387-389
exposing Objective-C classes to Swift, 394
extending Objective-C classes with Swift, 394-395
housecleaning, 394
running app, 396-399
updating classes in storyboard, 395-396
internal access, 248
internal keyword, 248
internal parameter names, 119
intersect, 53
introducers
let, 22, 79
var, 22, 79, 281
introspection, 255
iOS playgrounds, 6
isa pointers, 176
isDisjointWith() method, 54
isEmpty property, 22, 46
is-equal operator (==), 415
is keyword, 253, 291
isSubsetOf() method, 53
isSupersetOf() method, 53
iteration
forEach function, 406-409, 419
for-in loops, 97-100
map function, 404-408, 419
@interface statement (Objective-C), 378

J-K

JSON (JavaScript Object Notation), 257

key-value pairs, 47-48

keywords
Any, 256-257
AnyObject, 256-257
as, 255-256
as!, 291
Bool, 20
break, 65, 71
class, 239
convenience, 211, 308
defer, 122, 371-374
deinit, 262
didSet, 228, 233, 236
Double, 18-19
extension, 302
fallthrough, 71-72
final, 174, 231
get, 225-226, 281
guard, 371
if, 59-64
C/Objective C versus Swift, 60
else-if syntax, 61-63
if-else syntax, 60-63
if let, 80-81
ternary conditional operators, 63-64
@implementation, 381
import, 6
in, 135
init, 200, 208, 383
instancetype, 380
Int, 17-18
@interface, 378
internal, 248
is, 253, 291
lazy, 223
let, 7, 22, 79
mutating, 158, 283, 305
nesting, 74
nil, 77
objc, 298
optional, 290
override, 169, 253, 282
private, 248
protocol, 279
public, 248
rethrows, 345
self, 153, 203, 211, 242, 357
set, 226, 281
static, 239-241, 283
String, 20-22
strong, 379
super, 172, 231
switch
break statement, 71
explicit break, 65
fall through, 65
fallthrough statement, 71-72
matching multiple values, 65-66
range matching, 66-68
syntax, 64-65
tuple matching, 68-69
where keyword, 69-70
throws, 345, 365-366
try, 367-369
try!, 369-371, 375
try?, 370-371, 375
typealias, 246-247, 304, 337
UInt, 17
unowned, 270-272
var, 7, 22, 79, 281
weak, 269
where, 69-70, 352-353
willSet, 228, 233, 236

L

last property, 46
lazy keyword, 223
lazy stored properties, 223-224, 236
leaks, memory, 276
length of functions, 108
less than (<) operator, 31, 415
less than or equal to (<=) operator, 31
let keyword, 7, 22, 79
Library class, 320
Liskov Substitution Principle, 357
lists
closure capture lists, 272-275
tasks list, removing items from, 45
literals, 19
logical operators
combining, 35
explained, 33
logical AND, 33-34
logical NOT, 35
logical OR, 34-35
loops
for loops, 94
for-condition-increment loops, 94-95
for-in loops, 96-101
infinite loops, 91
transferring control
break statement, 103
continue statements, 102-103
while loops, 89-90
conditions, 90
repeat-while loops, 90-94
while loops, 90-91
maximum ranges of integers, finding, 17
mean, functions returning, 116-117
Media class, 286
median, functions returning, 116-117
median function
playgrounds, 142
rewriting, 139-140
MelonType enum, 204
memberwise initialization, 151, 155, 202-203
memory
deallocating, 261
memory leaks, 276
pointers, 22
memory leaks, 276
methods, 148. See also functions
adding with extensions, 305-306
append(), 43
area(), 175
arrays, 46
buttonTapped(), 293
calling, 322-324
classes, 155-156
configureView(), 393-395
doctor types, finding, 20-21
deinit, 262
description(), 177
dictionaries, 50
exclusiveOr(), 54
factory methods, 242
getters, 225-226, 379
init(), 48, 81, 210. See also initialization

M

Mac playgrounds, 6
manipulating arrays
adding elements to, 43
inserting elements into, 43-44
removing items from, 45-46
replacing items in, 44-45
updating, 44-45
mapEachElement, 134
map function, 404-408, 419
matching
advanced pattern matching, 72-73
for-in loops, 101-102
ranges, 66-68
tuples, 68-69
MathFunction class, 273
Mavericks, Xcode compatibility, 11
null-resettable attribute 433

MKA Annotation protocol, 290
mode, functions returning, 116-117
modifiers, @objc, 290
module bridging (Objective-C), 384-385
modulo operator (%), 29-30
Mountain Lion, Xcode compatibility, 11
multiple bindings, validating user input with, 85-86
multiple optionals, unwrapping, 83-84
multiple protocols adopting, 285-290
inheritance, 285-290
multiplication (*) operator, 29
mutability, 22
mutating functions, 283
mutating keyword, 158, 283, 305

N

named parameters, 190
naming conventions, 150
constants, 15
external parameter names in initializers, 206-207
parameters, 236
protocols, 284-285, 347
shorthand argument names, 138
variables, 15
Xcode projects, 4
nested functions, 130-132

nesting
method calls, 383
statements, 74
types, 309-310
nil coalescing operator, 83
nil keyword, 77
nil values explained, 77-78
forced unwrapping, 80, 87
Objective-C
nullable method parameters, 383
nullable property definitions, 382
None, 192
nonnull attribute, 382-383
non-specific types, 256-257
NOT operator, 28, 35
NSArray class, 79, 386
NSDictionary class, 47, 79, 387
NSError class, 364
NSHipster, 417
NSMutableDictionary class, 47
NSNumber class, 386
NSString class, 386
nullability
nil values explained, 77-78
forced unwrapping, 80, 87
Objective-C
nullable method parameters, 383
nullable property definitions, 382
_Nullable attribute, 383
null_resettable attribute, 382
Null_unspecified attribute, 383
numberOfSectionsInTableView method, 358
numeric types, 386

@objc modifier, 290
ObjCInteropTests folder, 399
objc keyword, 298
Objective-C
  allocation, 383-384
  bridging
definition of term, 384
module bridging, 384-385
type bridging, 385-387
  classes
  exposing to Swift, 394
  extending with Swift, 394-395
  compared to Swift, 22
  file structure
explained, 378
header files, 378-380
implementation files, 380-382
initialization, 383-384
  in-out parameters, 121
  integrating Swift into
  creating Swift class and bridging header, 389-393
downloading starter project, 387-389
  exposing Objective-C classes to Swift, 394
  extending Objective-C classes with Swift, 394-395
  housecleaning, 394
  running app, 396-399
  updating classes in storyboard, 395-396
  nested method calls, 383
  nullability
  nullable method parameters, 383
  nullable property definitions, 382
  overview, 377
  objects
  flattening
  flatMap function on optionals, 412-414
  flatMap on sequences, 411
  SharePoint Object Model, 318
  observers (property)
  explained, 228-230
  inheriting, 233-235
  overriding, 233-235
  Open/Closed Principle, 302
  open for extension, 301
  operands, 29
  operator functions, 139
  operators
as?, 291
  binary operators
  assignment operator, 14, 30, 44
  combinations of logical operators, 35
  comparison operators, 31-32
  compound assignment operators, 30-31
  explained, 29
  logical AND operator, 33-34
  logical NOT operator, 35
  logical OR operator, 34-35
  range operators, 32-33, 36
  remainder operator, 29-30
  standard arithmetic operators, 29
+ (concatenation), 9
  custom operators, 416-419
  identical to operator, 159
  infix operators, 29
  optional chaining operator (?), 318
  overloading, 414
  custom operators, 416-419
  with protocols, 415-416
  overview, 27
  ternary conditional operators, 35, 63-64
  type cast operator, 255-256
  type check operator, 253-255
  unary operators
decrement operator (\(-\)), 28
explained, 27
increment operator (\(++\)), 28
logical NOT operator, 28
prefix versus postfix, 27, 36
optional binding, 80-81
unwrap operator, 80
optional variables, 24, 265

order of initialization
(Objective-C), 382
Organization Identifier, 4
OR operator, 34-35
OS X support for Xcode, 11
out of bounds errors, 43
overloading
operators, 414
custom operators, 416-419
with protocols, 415-416
subscripts, 252-253
override keyword, 169, 253, 282

overrides, 168
default protocol implementations, 349-352
functions, 357
inherited methods, 169-172
instance methods, 170-171
preventing overrides, 231
property accessors, 231-233
property observers, 233-235
subscripts, 253

P

Paper struct, 250
parameters
closures, 137
default initializer parameters, 207-208
default parameter values, 120
external parameter names, 118-120, 206-207

function parameters,
111-112, 132-134
in-out parameters, 120-121
internal parameter names, 119
named parameters, 190
naming, 236
variadic parameters, 113-114

parent classes, 167
parentheses, 64
pattern matching. See matching
Person class, 267
Phone class, 266, 269-270
playgrounds, 6-8
creating, 6
Mac versus iOS
playgrounds, 6
median function, 142

plus sign (+), 29, 379
pointers, 22
postfix unary operators, 27, 36
prefix unary operators, 27, 36
prepareForSegue:sender:
method, 393

preventing overrides,
174-176, 231
print() function, 7, 329
printWeather() function, 328
private access, 248
private interfaces, 378
private keyword, 248
ProcessDelegate protocol, 372
program flow, controlling with
conditionals
advanced pattern matching, 72-73
program flow, controlling with conditionals

if statement, 59-64
  C/Objective C versus Swift, 60
  else-if syntax, 61-63
  if-else syntax, 60-63
  ternary conditional operators, 63-64
overview, 59
switch statement
  break statement, 71
  explicit break, 65
  fall through, 65
  fallthrough statement, 71-72
  matching multiple values, 65-66
  range matching, 66-68
  syntax, 64-65
  tuple matching, 68-69
  where keyword, 69-70
ternary conditional operators, 63-64

projects
  creating, 2-6
  naming, 4
  removing, 8

properties, 149, 221
  accessors
    getters, 225-226
    inheriting, 231-233
    overriding, 231-233
    overview, 225
    setters, 226-228
  arrays, 46
  classes, 155-156
  comparing, 150-151
  computed properties, 186, 224-225, 303-305
  dictionaries, 50
  finding, 20-21
  initializing in structs, 151
  nullable property definitions (Objective-C), 382
  optional chaining, 316-318
  property accessors, 231-233
  property observers
    explained, 228-230
    inheriting, 233-235
    overriding, 233-235
  protocol properties
    defining, 281-282
    optional properties, 290-291
  stored properties
    declaring, 221-222
    definition of term, 221
    instance variables, 222
    lazy stored properties, 223-224, 236
  type properties
    accessing, 239-241
    compared to instance properties, 258
  computed type properties, 241-242

protocol keyword, 279

protocols
  adoption
    adding with extensions, 310-312
    definition of term, 279
  multiple protocols, 285-290
  syntax, 280-281
  advantages of, 298
  associated types, 337-339
  ButtonDelegateProtocol, 293
  Comparable, 415, 419
  composition, 289-290
  converting protocol sequences, 355-357
  creating, 280-281
  definition of term, 279-280
  delegation, 292-297
  Equatable, 415
  ErrorType, 364-365, 375
  explained, 343
  extensions, 280
  GameManagerDelegate, 295
  HashableDelegate, 295
  Healable, 294
  heterogeneous collections, 353-355
  homogeneous collections, 353-355
  implementing, 344
  inheritance, 298
  methods
    default implementation, 298
    defining, 282-284
    optional methods, 290-291
  MKAnnotation, 290
  multiple protocols
    adopting, 285-290
    inheritance, 285-290
    naming conventions, 284-285, 347
    Optical, 286
random number generators, 93
range operators, 32-33, 36
ranges
  for-in loop, 97
  range operators, 32-33, 36
  switch statement, 66-68
raw values
  enums, 185-186
  getting, 186
  setting, 186-187
  initializer, 187
Ray, John, 6
Read-Eval-Print-Loop (REPL), 8-10
Rectangle class, 214
reduce function, 136
reduce instance method, 136
reference cycles
  closures, 272-275
  resolving, 269-272
  unowned references, 270-272
  weak references, 269-270
reinitializing, 261-265
reference count, 262
reference cycles
  identifying, 266-269
  resolving, 269-272
relationships, 265-266
strong references, 261, 265
versus values, 131
weak references, 269-270
relationships, reference, 265-266
remainder operator (%), 29-30
removeAll() method, 45
removeAtIndex() method, 45
removeLast() method, 45
removing
  array items, 45-46
  set elements, 52-53
  tasks list items, 45
  Xcode projects, 8
Rentable protocol
  creating, 280-281
  methods, 282-284
STYCustomStringConvertible, 395
public access, 247-248
public interface files (Objective-C), 378
public keyword, 248
pyramid of doom, 83-84
Pythagorean Theorem, 151

Q
question mark (?), 78, 208
Queue struct
  creating, 334-336
  extending, 336-337

R
overloading operators with, 415-416
overriding default protocol implementations, 349-352
ProcessDelegate, 372
properties
  defining, 281-282
  optional properties, 290-291
protocol composition, 289-290
protocol conformance, 279, 291
protocol extensions
  CollectionType, 345
  creating, 346-349
  explained, 344-345
  type constraints, 352-353
Rentable
  creating, 280-281
  methods, 282-284
STYCustomStringConvertible, 395
public access, 247-248
public interface files (Objective-C), 378
public keyword, 248
pyramid of doom, 83-84
Pythagorean Theorem, 151
rethrows keyword, 345
retrieving set elements, 52
returning functions
  higher order functions, 128-130
  syntax for, 130
returns someValue, 109
return statement, 115
return types, 109, 114-118
return values, ignoring, 118
rewriting median function, 139-140

S
safety, 212
Sams Teach Yourself iOS Application Development in 24 Hours (Ray), 6
scope, 111
self.init() method, 211
self keyword, 153, 203, 211, 242, 357
semicolon (;), 22
sequences
  flattening, 411
  protocol sequences, converting, 355-357
SequenceType extension, 405
servers, SharePoint Server, 318
set keyword, 226, 281
sets
  adding elements to, 51-52
  creating, 51
  definition of term, 51
  performing actions on, 54
  removing elements from, 52-53
  retrieving elements from, 52
  use case, 56
  when to use, 53-55
setters, 226-228, 379
SharePoint Object Model, 318
SharePoint Server, 318
shorthand syntax
  argument names, 138
  enums, 187-189
  single-expression closures, 137
  single inheritance, 172
Site class, 319
SOLID software development principles, 357
Some, 192
SomeClassWithExpensiveOperation class, 223
SomeClassWithLazyVar class, 223
Song class, 395
sorting dictionaries, 100
Square class, 215, 232
Square struct, 226-227
StackOfPapers class, 250-251
standard arithmetic operators, 29
starter project (Objective-C), downloading, 387-389
statements. See also keywords
  break, 65, 71
  defer, 122, 371-374
  fallback, 71-72
if, 59-64
  C/Objective C versus Swift, 60
  else-if syntax, 61-63
  if-else syntax, 60-63
  if let, 80-81
  ternary conditional operators, 63-64
import Cocoa, 6
import UIKit, 6
@interface, 378
nesting, 74
switch
  break statement, 71
  explicit break, 65
  fall through, 65
  fallthrough statement, 71-72
  matching multiple values, 65-66
  range matching, 66-68
  syntax, 64-65
  tuple matching, 68-69
  where keyword, 69-70
static dispatch, 349
static keyword, 239-241, 283
stored properties
  declaring, 221-222
  definition of term, 221
  instance variables, 222
  lazy stored properties, 223-224, 236
storyboard, updating classes in, 395-396
strings, 20-22
  combining, 9
  concatenating, 21
extending
   with computed properties, 304
   with subscripts, 306-307
for-in loops, 97-98
mutability, 22
string interpolation, 21-22
typebridginginObjective-C, 386
Stringtype. Seestrings
strong keyword, 379
strongly typed languages, 256
strong reference cycles, 261, 265
closures, 272-275
identifying, 266-269
resolving, 269-272
unowned references, 270-272
weak references, 269-270
structs, 20, 147, 148
adding, 152-153
AudioCD, 284-285
BluePotion, 294
comparing to classes, 149-150, 155-156
class instance references, 159-160
differences, 157-158
instance equality, 160
instance methods, 151-154
mutating properties, 158
properties, 150-151
deinitializers and, 276
DVD, 284-285
Enemy, 294
Heart, 294
initialization, 204-206
custominitializers, 203
initializationdelegation, 210-211
memberwise initialization, 202-203
memberwise initialization, 155
Paper, 250
Queue
   creating, 334-336
   extending, 336-337
Square, 226-227
syntax, 155
Volume, 244
when to use, 160-161
STYCustomStringConvertible
protocol, 395
stYMap function, 345
subclasses, 167
   creating, 167-168
   overriding instance methods, 170-171
subscript method, 251
subscripts, 166
   adding with extensions, 306-307
arrays, 41-42
creating, 249-252
optional chaining, 318-322
overloading, 252-253
overriding, 253
subtraction (-) operator, 29
super, accessing, 172-174
superclasses, 167
   overriding, 169
   preventing overrides, 174-175
super.init() method, 218
super keyword, 172, 231
TheSwiftProgramming
Language, 212
switching enum values, 190-192
switch statement, 190-191
   break statement, 71
   explicit break, 65
   fall through, 65
   fallthrough statement, 71-72
   matching multiple values, 65-66
   range matching, 66-68
   syntax, 64-65
tuple matching, 68-69
where keyword, 69-70

T
taskslist,removingitems
   from, 45
ternaryconditionaloperators,
   35, 63-64
TestAppproject, 2-6
TestClass, 263
TheLegendOfZeldaclass, 295
thread safe, 160
throwing errors, 365-366
throwing functions, catching
   errors from, 367-369
throws keyword, 345, 365-366
toInt() method, 23

tools, xcrun, 8

trailing closures, 139, 142-143

transferring

  control of execution
  break statement, 71
  fallthrough statement, 71-72
  control of loops
  with break statement, 103
  with continue statements, 102-103

try keyword, 367-369

try! keyword, 369-371, 375

try? keyword, 370-371, 375

tuples, 55-56, 115

  for-in loops, 101
  matching, 68-69

turnItDown() method, 245

type aliasing, 246-247

typealias keyword, 246-247, 304, 337

type casting

  definition of term, 253-257
  determining an instance’s type, 253-255
  downcasting, 255-256
  introspection, 255
  non-specific types, 256-257

type cast operator, 255-256

type check operator, 253-255

type inference, 15-16, 48-49, 97, 137

  typeMethodName() method, 242

  types, 13, 16. See also extensions

    access control, 247-248, 258
    annotation, 16

arrays, 332

  accessing values in, 41-42
  appending elements to, 43
  compared to dictionaries, 56
  creating, 41-42
  declaring, 39-41
  for-in loops, 98-99
  for loops, 96
  functions as members of arrays, 133
  generic function to return an array, 331
  indexes, 41-42
  inserting elements into, 43-44
  member types, 40
  methods, 46
  properties, 46
  removing items from, 45-46
  replacing items in, 44-45
  subscripts, 41-42
  tuples, 55-56
  type bridging in Objective-C, 386
  updating items in, 44-45
  zero-based arrays, 41

  associated types, 337-339
  Bool, 20, 399
  constraints, 329-332, 352-353
  declaring explicitly, 24
  Dictionary, 332
  Double, 18-19
  enums, 47, 183-184, 204-206
  associated values, 189-190
  creating, 185-188, 192-194
  custom initializers, 203-204
  initialization, 204-206
  raw values, 185-187
  structure of, 184-185
  switching values, 190-192
  syntax, 187-189

  error types, 364-365

  finding properties/methods of, 20-21

  functions, 110-111

generic types

  creating, 333-336
  extending, 336-337
  mixing with non-generic parameters, 340
  multiple generic types, 340
  queues, 334-337

  id, 79, 256

initialization, 201

  class types, 207-209
  convenience initializers, 211
  default initializers, 201
  definition of term, 199
  designated initializers, 211
  goal of initializers, 200-201
  inherited initializers, 211
  initialization delegation, 209-212
user input, validating

unary operators

decrement operator (\(-\)), 28
explained, 27
increment operator (\(++\)), 28
logical NOT operator, 28
prefix versus postfix, 27, 36
unary minus operator, 29
unary plus operator, 29

unary plus operator, 29
underscore (_), 68, 206, 383
unions, 53
unique function, 331
unowned keyword, 270-272
unowned references, 270-272, 276
unsigned integers, 17
unwrap operator, 80-81
unwrapping optionals, 23
explained, 79-80
forced unwrapping, 80
implicitly unwrapped optionals, 81-82
multiple optionals, 83-84
nil coalescing operator, 83
optional binding, 80-81
unwrap operator, 80

updateValue() method, 49-50

updating
array elements, 44-45
classes in storyboard, 395-396

use cases
optional chaining, 316
optionals, 85-87
sets, 56

user input, validating, 85-86
validation

user input, 85-86
value validation, 112

values

accessing in arrays, 41-42
binding, 68
comparing, 92-93
constants
assigning values to, 14
explained, 13-14
naming, 15
when to use, 14
copying, 149
default property value, 200
default values, 202-206
enumerations, 47
explained, 78
filtering, 409-410
initialization, 22-23, 41, 201
external parameter names
in initializers, 206-207
setting default values, 202-206
literals, 19
nil values
explained, 77-78
forced unwrapping, 80, 87
Objective-C, 382-383
optional values, 28
additional reading, 84
designating variables as, 78-79
explained, 77-78

flattening, 412-414
pyramid of doom, 83-84
use case, 85-87
wrapping/unwrapping, 23, 79-82
struct, 20
value binding, 68
variables, 14-15
versus references, 131

value types, 148
value validation, 112
variables
creating, 79
declaring, 14-15, 22-23
designating as optionals, 78-79
explained, 14-15
instance variables, 222
naming, 15
optionals
additional reading, 84
designating, 78-79
explained, 77-78

variadic parameters, 113-114
var keyword, 7, 22, 79, 281
verifying protocol conformance, 291
view controllers, 173-174
viewDidLoad() method, 174, 393
Volume struct, 244

weak keyword, 269
weak references, 269-270

websites
Apple documentation, 417
NSHipster, 417
Welcome to Xcode screen, 3
where keyword, 69-70, 87, 352-353
while loops, 89-91
conditions, 90
repeat-while loops, 90-94
repeat-while loops, 92-93
wildcards, 68, 101
willSet keyword, 228, 233, 236
wrappers, 78
wrapping optionals, 79-80

X-Y-Z

Xcode
Auto Complete, 394
downloading, 2
playgrounds, 6-8
projects
creating, 2-6
naming, 4
removing, 8
raw value initializer, 187
xcrun, 8

Yosemite, support for Xcode, 11

zero-based arrays, 41