Dedication

For my wife, who is smart, strong, and virtuous. And for my family, who has given me every opportunity to live a good life.

— M.M.

For my wife and best friend; you are “s’wonderful.” And for my daughters, who bring me joy every day.

— J.G.
This page intentionally left blank
Acknowledgments

We received a lot of help in writing this book. Without it, this book would not be what it is, and it may have never even happened. Thanks are due.

First, we need to say thank you to our colleagues at Big Nerd Ranch. Thank you to Aaron Hillegass for providing us with the opportunity to write this book. It has been immensely gratifying to learn and teach Swift. Big Nerd Ranch provided us with the time and space to work on this project. We hope that this book lives up to the trust and the support that we have received.

Particular thanks are also due to our colleagues in the Cocoa Pod at Big Nerd Ranch. Your careful teaching revealed many bugs in the text, and your thoughtful recommendations led to many improvements in our approach. Those of you who are not instructors helped to review the materials, vetted our approach, and provided countless suggestions that we never thought of. It is truly wonderful to have colleagues such as you. Thank you Pouria Almassi, Matt Bezark, Nate Chandler, Step Christopher, Kynerd Coleman, Matthew Compton, Joseph Dixon, Robert Edwards, Sean Farrell, Brian Hardy, Florian Harr, Tom Harrington, Bolot Kerimbaev, Christian Keur, JJ Manton, Bill Monk, Chris Morris, Adam Preble, Scott Richie, Jeremy Sherman, Steve Sparks, Rod Strougo, TJ Usiyan, Zach Waldowski, Thomas Ward, and Mike Zornek.

Our colleagues in operations and sales are instrumental. Classes would literally never be scheduled without their work. Thank you Shannon Coburn, Nicole Rej, Heather Brown, Tasha Schroader, Mat Jackson, and Chris Kirksey for all of your hard work. We cannot do what you do.

Second, we need to acknowledge the many talented folks who worked on the book with us.

Elizabeth Holaday, our editor, helped refine the book, crystallize its strengths, and diminish its weaknesses.

Simone Payment, our copy-editor, found and corrected errors and ultimately made us look smarter than we are.

Ellie Volckhausen designed our cover; that skateboard looks pretty rad.

Chris Loper designed and produced the print book and the EPUB and Kindle versions.

Finally, thank you to our students. We learned with you and for you. Teaching is part of the greatest thing that we do, and it has been a pleasure working with you. We hope that the quality of this book matches your enthusiasm and determination.
This page intentionally left blank
# Table of Contents

Introduction .................................................................................................................... xv  
Learning Swift ........................................................................................................ xv  
Whither Objective-C? ............................................................................................... xv  
Prerequisites .......................................................................................................... xv  
How This Book Is Organized ................................................................................... xvi  
How to Use This Book ............................................................................................ xvi  
Challenges ............................................................................................................. xvii  
For the More Curious ............................................................................................. xvii  
Typographical Conventions .................................................................................. xvii  
Necessary Hardware and Software ........................................................................ xviii  
Before We Begin .................................................................................................. xviii  

I. Getting Started ........................................................................................................ 1  

1. Getting Started ...................................................................................................... 3  
   Getting Started with Xcode ................................................................................. 3  
   Playing in a Playground ..................................................................................... 6  
   Varying Variables and Printing to the Console .................................................... 7  
   You Are On Your Way! ...................................................................................... 9  
   Bronze Challenge .............................................................................................. 9  

2. Types, Constants, and Variables ........................................................................... 11  
   Types ............................................................................................................. 11  
   Constants vs. Variables ..................................................................................... 13  
   String Interpolation .......................................................................................... 14  
   Bronze Challenge ............................................................................................ 15  

II. The Basics ............................................................................................................. 17  

3. Conditionals ........................................................................................................ 19  
   if/else ............................................................................................................ 19  
   Ternary Operator ............................................................................................. 22  
   Nested ifs ..................................................................................................... 23  
   else if ........................................................................................................... 24  
   Bronze Challenge ........................................................................................... 24  

4. Numbers ............................................................................................................. 25  
   Integers .......................................................................................................... 25  
   Creating Integer Instances ................................................................................. 27  
   Operations on Integers ...................................................................................... 28  
   Integer division ............................................................................................... 29  
   Operator shorthand .......................................................................................... 30  
   Overflow operators ....................................................................................... 30  
   Converting Between Integer Types ...................................................................... 32  
   Floating-Point Numbers .................................................................................... 33  
   Bronze Challenge ........................................................................................... 34  

5. Switch ................................................................................................................ 35  
   What Is a Switch? ............................................................................................ 35  
   Switch It Up ................................................................................................... 36  
   Ranges ............................................................................................................ 39  
   Value binding ................................................................................................. 40
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>III. Collections and Functions</td>
<td></td>
</tr>
<tr>
<td>6. Loops</td>
<td>47</td>
</tr>
<tr>
<td>for-in Loops</td>
<td>47</td>
</tr>
<tr>
<td>for case</td>
<td>50</td>
</tr>
<tr>
<td>A Quick Note on Type Inference</td>
<td>51</td>
</tr>
<tr>
<td>for Loops</td>
<td>51</td>
</tr>
<tr>
<td>while Loops</td>
<td>52</td>
</tr>
<tr>
<td>repeat-while Loops</td>
<td>53</td>
</tr>
<tr>
<td>Control Transfer Statements, Redux</td>
<td>54</td>
</tr>
<tr>
<td>Bronze Challenge</td>
<td>56</td>
</tr>
<tr>
<td>7. Strings</td>
<td>57</td>
</tr>
<tr>
<td>Working with Strings</td>
<td>57</td>
</tr>
<tr>
<td>Unicode</td>
<td>59</td>
</tr>
<tr>
<td>Unicode scalars</td>
<td>59</td>
</tr>
<tr>
<td>Canonical equivalence</td>
<td>61</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>63</td>
</tr>
<tr>
<td>8. Optionals</td>
<td>65</td>
</tr>
<tr>
<td>Optional Types</td>
<td>65</td>
</tr>
<tr>
<td>Optional Binding</td>
<td>67</td>
</tr>
<tr>
<td>Implicitly Unwrapped Optionals</td>
<td>69</td>
</tr>
<tr>
<td>Optional Chaining</td>
<td>70</td>
</tr>
<tr>
<td>Modifying an Optional in Place</td>
<td>71</td>
</tr>
<tr>
<td>The Nil Coalescing Operator</td>
<td>71</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>72</td>
</tr>
<tr>
<td>III. Collections and Functions</td>
<td>73</td>
</tr>
<tr>
<td>9. Arrays</td>
<td>75</td>
</tr>
<tr>
<td>Creating an Array</td>
<td>75</td>
</tr>
<tr>
<td>Accessing and Modifying Arrays</td>
<td>77</td>
</tr>
<tr>
<td>Array Equality</td>
<td>83</td>
</tr>
<tr>
<td>Immutable Arrays</td>
<td>84</td>
</tr>
<tr>
<td>Documentation</td>
<td>85</td>
</tr>
<tr>
<td>Bronze Challenge</td>
<td>86</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>86</td>
</tr>
<tr>
<td>10. Dictionaries</td>
<td>87</td>
</tr>
<tr>
<td>Creating a Dictionary</td>
<td>87</td>
</tr>
<tr>
<td>Populating a Dictionary</td>
<td>88</td>
</tr>
<tr>
<td>Accessing and Modifying a Dictionary</td>
<td>88</td>
</tr>
<tr>
<td>Adding and Removing Values</td>
<td>90</td>
</tr>
<tr>
<td>Looping</td>
<td>92</td>
</tr>
<tr>
<td>Immutable Dictionaries</td>
<td>93</td>
</tr>
<tr>
<td>Translating a Dictionary to an Array</td>
<td>93</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>94</td>
</tr>
<tr>
<td>11. Sets</td>
<td>95</td>
</tr>
<tr>
<td>What Is a Set?</td>
<td>95</td>
</tr>
<tr>
<td>Getting a Set</td>
<td>95</td>
</tr>
</tbody>
</table>
Swift Programming

### Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working with Sets</td>
<td>97</td>
</tr>
<tr>
<td>Unions</td>
<td>97</td>
</tr>
<tr>
<td>Intersects</td>
<td>98</td>
</tr>
<tr>
<td>Disjoint</td>
<td>99</td>
</tr>
<tr>
<td>Bronze Challenge</td>
<td>100</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>100</td>
</tr>
<tr>
<td>12. Functions</td>
<td>101</td>
</tr>
<tr>
<td>A Basic Function</td>
<td>101</td>
</tr>
<tr>
<td>Function Parameters</td>
<td>102</td>
</tr>
<tr>
<td>Parameter names</td>
<td>103</td>
</tr>
<tr>
<td>Variadic parameters</td>
<td>104</td>
</tr>
<tr>
<td>Default parameter values</td>
<td>105</td>
</tr>
<tr>
<td>In-out parameters</td>
<td>106</td>
</tr>
<tr>
<td>Returning from a Function</td>
<td>107</td>
</tr>
<tr>
<td>Nested Functions and Scope</td>
<td>108</td>
</tr>
<tr>
<td>Multiple Returns</td>
<td>108</td>
</tr>
<tr>
<td>Optional Return Types</td>
<td>110</td>
</tr>
<tr>
<td>Exiting Early from a Function</td>
<td>111</td>
</tr>
<tr>
<td>Function Types</td>
<td>111</td>
</tr>
<tr>
<td>Bronze Challenge</td>
<td>112</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>112</td>
</tr>
<tr>
<td>13. Closures</td>
<td>113</td>
</tr>
<tr>
<td>Closure Syntax</td>
<td>113</td>
</tr>
<tr>
<td>Closure Expression Syntax</td>
<td>115</td>
</tr>
<tr>
<td>Functions as Return Types</td>
<td>117</td>
</tr>
<tr>
<td>Functions as Arguments</td>
<td>118</td>
</tr>
<tr>
<td>Closures Capture Values</td>
<td>121</td>
</tr>
<tr>
<td>Closures Are Reference Types</td>
<td>122</td>
</tr>
<tr>
<td>Functional Programming</td>
<td>123</td>
</tr>
<tr>
<td>Higher-order functions</td>
<td>123</td>
</tr>
<tr>
<td>Gold Challenge</td>
<td>126</td>
</tr>
<tr>
<td>IV. Enumerations, Structures, and Classes</td>
<td>127</td>
</tr>
<tr>
<td>14. Enumerations</td>
<td>129</td>
</tr>
<tr>
<td>Basic Enumerations</td>
<td>129</td>
</tr>
<tr>
<td>Raw Value Enumerations</td>
<td>132</td>
</tr>
<tr>
<td>Methods</td>
<td>135</td>
</tr>
<tr>
<td>Associated Values</td>
<td>138</td>
</tr>
<tr>
<td>Recursive Enumerations</td>
<td>141</td>
</tr>
<tr>
<td>Bronze Challenge</td>
<td>144</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>144</td>
</tr>
<tr>
<td>15. Structs and Classes</td>
<td>145</td>
</tr>
<tr>
<td>A New Project</td>
<td>145</td>
</tr>
<tr>
<td>Structures</td>
<td>150</td>
</tr>
<tr>
<td>Instance Methods</td>
<td>153</td>
</tr>
<tr>
<td>Mutating methods</td>
<td>154</td>
</tr>
<tr>
<td>Classes</td>
<td>155</td>
</tr>
<tr>
<td>A monster class</td>
<td>155</td>
</tr>
<tr>
<td>Inheritance</td>
<td>156</td>
</tr>
<tr>
<td>Strong Reference Cycles</td>
<td>290</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Reference Cycles in Closures</td>
<td>295</td>
</tr>
<tr>
<td>Bronze Challenge</td>
<td>298</td>
</tr>
<tr>
<td>Silver Challenge</td>
<td>298</td>
</tr>
<tr>
<td>For the More Curious: Can I Retrieve the Reference Count of an Instance?</td>
<td>299</td>
</tr>
</tbody>
</table>

25. Equatable and Comparable

| Conforming to Equatable | 301 |
| Conforming to Comparable | 304 |
| Comparable’s Inheritance | 307 |
| Bronze Challenge | 307 |
| Gold Challenge | 307 |
| Platinum Challenge | 308 |
| For the More Curious: Custom Operators | 308 |

VI. Event-Driven Applications

26. Your First Cocoa Application

| Getting Started with VocalTextEdit | 313 |
| Model-View-Controller | 315 |
| Setting Up the View Controller | 317 |
| Setting Up Views in Interface Builder | 320 |
| Adding the Speak and Stop buttons | 322 |
| Adding the text view | 323 |
| Auto Layout | 326 |
| Making Connections | 329 |
| Setting target-action pairs for VocalTextEdit’s buttons | 329 |
| Connecting the text view outlet | 330 |
| Making VocalTextEdit… Vocal | 331 |
| Saving and Loading Documents | 334 |
| Type casting | 337 |
| Saving documents | 337 |
| Loading documents | 339 |
| MVC cleanup | 342 |
| Silver Challenge | 344 |
| Gold Challenge | 344 |

27. Your First iOS Application

| Getting Started with iTahDoodle | 345 |
| Laying Out the User Interface | 346 |
| Wiring up your interface | 348 |
| Modeling a To-Do List | 358 |
| Setting Up the UITableView | 360 |
| Saving and Loading TodoList | 367 |
| Saving TodoList | 367 |
| Loading TodoList | 369 |
| Bronze Challenge | 370 |
| Silver Challenge | 370 |
| Gold Challenge | 370 |

28. Interoperability

| An Objective-C Project | 371 |
| Creating a contacts app | 374 |
Adding Swift to an Objective-C Project ........................................ 382
  Adding contacts ........................................................................ 386
Adding an Objective-C Class ......................................................... 396
  Silver Challenge ....................................................................... 402
  Gold Challenge ....................................................................... 402
29. Conclusion ............................................................................. 403
  Where to Go from Here? ............................................................ 403
  Shameless Plugs ...................................................................... 403
  An Invitation ............................................................................ 403
Index .......................................................................................... 405
This page intentionally left blank
Introduction

Learning Swift

Apple’s World Wide Developers Conference is an annual landmark event for its developer community. It is a big deal every year, but 2014 was particularly special: Apple introduced an entirely new language called Swift for the development of iOS and OS X applications.

As a new language, Swift represents a fairly dramatic shift for Mac OS X and iOS developers. More experienced iOS developers have something new to learn, and new developers cannot rely on a venerable community for tried and true answers and patterns. Naturally, this shift creates some uncertainty.

But this is also an exciting time to be a Mac OS X and iOS developer. There is a lot to learn in a new language, and this is especially true for Swift. The language has evolved quite a bit since its beta release in the summer of 2014, and it continues to evolve.

We are all at the forefront of this language’s development. As new features are added to Swift, its users can collaboratively determine its best practices. You can directly contribute to this conversation, and your work with this book will start you on your way to becoming a contributing member of the Swift community.

Whither Objective-C?

So, what about Objective-C, Apple’s previous *lingua franca* for its platforms? Do you still need to know that language? For the time being, we think that answer is an unequivocal “Yes.” Apple’s Cocoa library, which you will use extensively, is written in Objective-C, so debugging will be easier if you understand that language. Moreover, most learning materials and existing Mac and iOS apps are written in Objective-C. Indeed, Apple has made it easy, and sometimes preferable, to mix and match Objective-C with Swift in the same project. As an iOS or Mac developer, you are bound to encounter Objective-C, so it makes sense to be familiar with the language.

But do you need to know Objective-C to learn Swift? Not at all. Swift coexists and interoperates with Objective-C, but it is its own language. If you do not know Objective-C, it will not hinder you in learning Swift. (We will only use Objective-C directly in one chapter toward the end of this book, and even then it will not be important for you to understand the language.)

Prerequisites

We have written this book for all types of iOS and Mac OS X developers, from platform experts to first-timers. For readers just starting software development, we will highlight and implement best practices for Swift and programming in general. Our strategy is to teach you the fundamentals of programming while learning Swift. For more experienced developers, we believe this book will serve as a helpful introduction to your platform’s new language. So while having some development experience will be helpful, we do not believe that it is necessary in order to have a good experience with this book.

We have also written this book with numerous examples so that you can refer to it in the future. Instead of focusing on abstract concepts and theory, we have written in favor of the practical. Our approach
favors using concrete examples to unpack the more difficult ideas and also to expose the best practices
that make code more fun to write, more readable, and easier to maintain.

How This Book Is Organized

This book is organized in six parts. Each is designed to accomplish a specific set of goals that build on
each other. By the end of the book, you will have built your knowledge of Swift from that of a beginner
to a more advanced developer.

Getting Started

This part of the book focuses on the tools that you will need
to write Swift code and introduces Swift’s syntax.

The Basics

The Basics introduces the fundamental data types that you
will use every day as a Swift developer. This part of the book
also covers Swift’s control flow features that will help you to
control the order in which your code executes.

Collections and Functions

You will often want to gather related data in your application.
Once you do, you will want to operate on that data. Swift
offers collections and functions to help with these tasks.

Enumerations, Structures, and
Classes

This part of the book covers how you will model your data
in your own development. We cover the differences between
these types and make some recommendations on when to use
each.

Advanced Swift

As a modern language, Swift provides a number of more
advanced features that enable you to write elegant, readable,
and effective code. This part of the book discusses how to use
these elements of Swift to write idiomatic code that will set
you apart from more casual Swift developers.

Event-Driven Applications

This part of the book walks you through writing your first
Mac OS X and iOS applications. For readers working with
older Mac OS X or iOS applications, we conclude this part
of the book by discussing how to interoperate between
Objective-C and Swift.

How to Use This Book

Programming can be tough, and this book is here to make it easier. How can we help you with that?
Follow these steps:

• Read the book. Really! Do not just browse it nightly before going to bed.

• Type out the examples as you read along. Part of learning is muscle memory. If your fingers know
where to go and what to type without too much thought on your part, then you are on your way to
becoming a more effective developer.
• Make mistakes! In our experience, the best way to learn how things work is to first figure out what makes them not work. Break our code examples and then make them work again.

• Experiment as your imagination sees fit. Whether that means tinkering with the code you find in the book or going off in your own direction, the sooner you start solving your own problems with Swift, the faster you will become a better developer.

• Do the challenges we have included in most chapters. As we mentioned, it is important to begin solving problems with Swift as soon as possible. Doing so will help you to start thinking like a developer.

More experienced developers may not need to go through some of the earlier parts of the book. Getting Started and The Basics may be very familiar to some developers.

One caveat: In The Basics, do not skip the chapter on Optionals as they are at the heart of Swift, and in many ways they define what is unique about the language.

Subsequent chapters like Arrays, Dictionaries, Functions, Enumerations, and Structs and Classes may seem like they will not present anything new to the practiced developer, but we feel that Swift’s approach to these topics is unique enough that every reader should at least skim these chapters.

Last, remember that learning new things takes time. Dedicate some time to going through this book when you are able to avoid distractions. You will get more out of the text if you can.

**Challenges**

Many of the chapters conclude with an exercise for you to work through on your own. These are an excellent opportunity for you to challenge yourself. In our experience, truly deep learning is accomplished when you solve problems in your own way.

**For the More Curious**

Relatedly, we include sections entitled “For the More Curious” at the end of many chapters. These sections address questions that may have occurred to the curious reader working through the chapter. Sometimes, we discuss how a given language feature’s underlying mechanics work, or we may explore a programming concept not quite related to the heart of the chapter.

**Typographical Conventions**

You will be writing a lot of code as you work through this book. To make things easier, we use a couple of conventions to identify what text is old, what should be added, and what should be removed. For example, in the function implementation below, you are deleting the text `print("Hello")` and adding `print("Goodbye")`.

```swift
func talkToMe() {
    print("Hello")
    print("Goodbye")
}
```
Necessary Hardware and Software

To build and run the applications in this book, you will need a Mac running OS X Yosemite (10.10) or newer. You will also need to install Xcode, Apple’s integrated development environment (IDE), which is available on the App Store. Xcode includes the Swift compiler as well as other development tools you will use throughout the book.

Swift is still under rapid development. This book is written for Swift 2.0 and Xcode 7.0. Many of the examples will not work as written if you are using an older version of Xcode. If you are using a newer version of Xcode, it is possible there may have been changes in the language that will cause some examples to fail.

As this book is moving into the printing process, Xcode 7.1 Beta is available. The code samples in the book work with the latest beta version we have been able to use. If future versions of Xcode do cause problems, take heart – the vast majority of what you learn will continue to be applicable to future versions of Swift even though there may be changes in syntax or names. You can also check out our forums at http://forums.bignerdranch.com for help.

Before We Begin

We hope to show you how much fun it can be to make applications for the Apple ecosystem. While writing code can be extremely frustrating, it can also be gratifying. There is something magical and exhilarating about solving a problem, not to mention the special joy that comes out of making an app that helps people and brings them happiness.

The best way to improve at anything is with practice. If you want to be a developer, then let’s get started! If you find that you do not think you are very good at it, who cares? Keep at it and we are sure that you will surprise yourself. Your next steps lie ahead. Onward!
This page intentionally left blank
In previous chapters your code led a relatively simple life: you declared some simple constants and variables and then assigned them values. But of course, an application really comes to life – and programming becomes a bit more challenging – when the application makes decisions based on the contents of its variables. For example, a game may let players leap a tall building if they have eaten a power-up. You use conditional statements to help applications make these kind of decisions.

**if/else**

if/else statements execute code based on a specific logical condition. You have a relatively simple either/or situation and depending on the result one branch of code or another (but not both) runs. Consider Knowhere, your small town from the previous chapter, and imagine that you need to buy stamps. Either Knowhere has a post office or it does not. If it has a post office, you will buy stamps there. If it does not have a post office, you will need to drive to the next town to buy stamps. Whether there is a post office is your logical condition. The different behaviors are “get stamps in town” and “get stamps out of town.”

Some situations are more complex than a binary yes/no. You will see a more flexible mechanism called switch in Chapter 5. But for now, let’s keep it simple.

Create a new OS X playground and name it **Conditionals**. Enter the code below, which shows the basic syntax for an if/else statement:

**Listing 3.1 Big or small?**

```swift
import Cocoa

var population: Int = 5422
var message: String

if population < 10000 {
    message = "\(population) is a small town!"
} else {
    message = "\(population) is pretty big!"
}

print(message)
```

You first declare `population` as an instance of the `Int` type and then assign it a value of 5422. Next, you declare a variable called `message` that is of the `String` type. You leave this variable uninitialized at first, meaning that you do not assign it a value.
Next comes the conditional `if/else` statement. This is where `message` is assigned a value based on whether the “if” statement evaluates to true. (Notice that you use *string interpolation* to put the population into the `message` string.)

Figure 3.1 shows what your playground should look like. The console and the results sidebar show that `message` has been set to be equal to the string literal assigned when the conditional evaluates to true. How did this happen?

**Figure 3.1 Conditionally describing a town’s population**

![Screenshot of Swift playground](attachment:image.png)

The condition in the `if/else` statement tests whether your town’s population is less than 10,000 via the `<` comparison operator. If the condition evaluates to true, then `message` is set to be equal to the first string literal (“X is a small town!”). If the condition evaluates to false – if the population is 10,000 or greater – the `message` is set to be equal to the second string literal (“X is pretty big!”). In this case, the town’s population is less than 10,000, so `message` is set to “5422 is a small town!”

Table 3.1 lists Swift’s comparison operators.

**Table 3.1 Comparison operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>Evaluates whether the number on the left is smaller than the number on the right.</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Evaluates whether the number on the left is smaller than or equal to the number on the right.</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Evaluates whether the number on the left is greater than the number on the right.</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Evaluates whether the number on the left is greater than or equal to the number on the right.</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Evaluates whether the number on the left is equal to the number on the right.</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Evaluates whether the number on the left is not equal to the number on the right.</td>
</tr>
<tr>
<td><code>===</code></td>
<td>Evaluates whether the two instances point to the same reference.</td>
</tr>
<tr>
<td><code>!==</code></td>
<td>Evaluates whether the two instances do not point to the same reference.</td>
</tr>
</tbody>
</table>
You do not need to understand all of the operators’ descriptions right now. You will see many of them in action as you move through the book, and they will become clearer as you use them. Refer back to this table as a reference if you have questions.

Sometimes you only care about one aspect of the condition that is under evaluation. That is, you want to execute code if a certain condition is met and do nothing if it is not. Enter the code below. (Notice that new code, shown in bold, appears in two places.)

**Listing 3.2 Is there a post office?**

```swift
import Cocoa

var population: Int = 5422
var message: String
var hasPostOffice: Bool = true

if population < 10000 {
    message = "\(population) is a small town!"
} else {
    message = "\(population) is pretty big!"
}

print(message)

if !hasPostOffice {
    print("Where do we buy stamps?")
}
```

Here, you add a new variable called `hasPostOffice`. This variable has the type `Bool`, short for “Boolean.” Boolean types can take one of two values: true or false. In this case, the Boolean `hasPostOffice` variable keeps track of whether the town has a post office. You set it to true, meaning that it does.

The `!` is called a **logical operator**. This operator is known as “logical not.” It tests whether `hasPostOffice` is false. You can think of `!` as inverting a **Boolean** value: true becomes false, and false becomes true.

The code above first sets `hasPostOffice` to true, then asks whether it is false. If `hasPostOffice` is false, you do not know where to buy stamps, so you ask. If `hasPostOffice` is true, you know where to buy stamps and do not have to ask, so nothing happens.

Because the town **does** have a post office (because `hasPostOffice` was initialized to true), the condition `!hasPostOffice` is false. That is, it is **not** the case that `hasPostOffice` is false. Therefore, the `print()` function never gets called.

Table 3.2 lists Swift’s logical operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&amp;</code></td>
<td>Logical AND: true if and only if both are true (false otherwise)</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>Logical OR: true if either is true (false only if both are false)</td>
</tr>
<tr>
<td><code>!</code></td>
<td>Logical NOT: true becomes false, false becomes true</td>
</tr>
</tbody>
</table>
Ternary Operator

The ternary operator is very similar to an if/else statement, but has more concise syntax. The syntax looks like this: `a ? b : c`. In English, the ternary operator reads something like, “If a is true, then do b. Otherwise, do c.”

Let’s rewrite the town population check that used if/else using the ternary operator instead.

Listing 3.3 Using the ternary operator

```swift
... if population < 10000 {
    message = "\(population) is a small town!"
} else {
    message = "\(population) is pretty big!"
} message = population < 10000 ? "\(population) is a small town!" : "\(population) is pretty big!"
...
```

The ternary operator can be a source of controversy: some programmers love it; some programmers loathe it. We come down somewhere in the middle. This particular usage is not very elegant. Your assignment to `message` requires more than a simple `a ? b : c`. The ternary operator is great for concise statements, but if your statement starts wrapping to the next line, we think you should use if/else instead.

Hit Command-Z to undo, removing the ternary operator and restoring your if/else statement.

Listing 3.4 Restoring if/else

```swift
... message = population < 10000 ? "\(population) is a small town!" : "\(population) is pretty big!"
if population < 10000 {
    message = "\(population) is a small town!"
} else {
    message = "\(population) is pretty big!"
}
...
Nested ifs

You can nest if statements for scenarios with more than two possibilities. You do this by writing an if/else statement inside the curly braces of another if/else statement. To see this, nest an if/else statement within the else block of your existing if/else statement.

Listing 3.5 Nesting conditionals

```swift
import Cocoa
var population: Int = 5422
var message: String
var hasPostOffice: Bool = true
if population < 10000 {
    message = "\(population) is a small town!"
} else {
    if population >= 10000 && population < 50000 {
        message = "\(population) is a medium town!"
    } else {
        message = "\(population) is pretty big!"
    }
}
print(message)
if !hasPostOffice {
    print("Where do we buy stamps?")
}
```

Your nested if clause makes use of the `>=` comparator (comparison operator) and the `&&` logical operator to check whether population is within the range of 10,000 to 50,000. Because your town’s population does not fall within that range, your message is set to “5422 is a small town!” as before.

Try bumping up the population to exercise the other branches.

Nested if/else statements are common in programming. You will find them out in the wild, and you will be writing them as well. There is no limit to how deeply you can nest these statements. However, the danger of nesting them too deeply is that it makes the code harder to read. One or two levels are fine, but beyond that your code becomes less readable and maintainable.

There are ways to avoid nested statements. Next, you are going to refactor the code that you have just written to make it a little easier to follow. Refactoring means changing code so that it does the same work but in a different way. It may be more efficient, or may just look prettier or be easier to understand.
else if

The else if conditional lets you chain multiple conditional statements together. else if allows you to check against multiple cases and conditionally executes code depending on which clause evaluates to true. You can have as many else if clauses as you want. Only one condition will match.

To make your code a little easier to read, extract the nested if/else statement to be a standalone clause that evaluates whether your town is of medium size.

Listing 3.6 Using else if

```swift
import Cocoa

var population: Int = 5422
var message: String
var hasPostOffice: Bool = true

if population < 10000 {
    message = "\(population) is a small town!"
} else if population >= 10000 && population < 50000 {
    message = "\(population) is a medium town!"
} else {
    if population >= 10000 && population < 50000 {
        message = "\(population) is a medium town!"
    } else {
        message = "\(population) is pretty big!"
    }
    message = "\(population) is pretty big!"
}

print(message)

if !hasPostOffice {
    print("Where do we buy stamps?")
}
```

You are using one else if clause, but you could have chained many more. This block of code is an improvement over the nested if/else above. If you find yourself with lots of if/else statements, you may want to use another mechanism, such as switch described in Chapter 5. Stay tuned.

Bronze Challenge

Add an additional else if statement to the town-sizing code to see if your town’s population is very large. Choose your own population thresholds. Set the message variable accordingly.
Index

Symbols
! (force-unwrap operator), 66
! (implicitly unwrapped optionals), 69, 201
!(not operator), 21
!= operator, 303
$0 (argument reference), 116
% operator, 109
%= operator, 30
&& operator, 23
&+ operator, 31
*= operator, 30
+ operator, 7
++ operator, 30
+= operator, 7, 30, 81
-- operator, 30
-= operator, 30
. syntax, 154
... syntax, 104
// (code comment), 6
/= operator, 30
: for protocol conformance, 229
< operator, 20, 304
<> syntax, 87, 266
= operator, 7, 11
== operator, 33, 83, 218, 302, 303
=== operator, 218
>= operator, 23
? (failable initializers), 201
? (optional), 65
@IBAction, 331
@IBOutlet, 331
[:] (Dictionary literal syntax), 88
[] (Array literal syntax), 76
`() (string interpolation), 15
\u{} syntax, 60
_(as parameter name), 206
_(wildcard), 44
|| operator, 21

A
a ? b : c statements, 22
access control, 180-183
action segues, 390
addition assignment operator (+=), 7, 30, 81
addition operator (+), 7
advancedBy(_:) function, 62
and operator (&&), 21
append(_:) function, 77
appendContentsOf(_:) method, 71
Application Programming Interfaces (APIs), 6
application sandbox, 367
applications, document-based, 313
ARC (Automatic Reference Counting), 289
arguments
(see also parameters)
functions as, 118-120
shorthand names for, 116
Array index out of range error, 340
Array literals, 76
Array() syntax, 93
arrays
about, 75
appending items, 77
changing items, 79
checking equality of, 83, 84
combining, 81, 82
creating sets from, 96
declaring, 75, 76
dumping, 125
immutable, 84, 85
initializing, 76
inserting items, 82
inserting over, 80, 81
mapping contents, 124
NSArray, 368
reducing, 126
removing items, 78, 80
sets vs., 95
sorting, 113-117
 subscripting, 79, 80
as! operator, 337
assert(_:_:) function, 241
assertions, 241
assignment operator (=), 7, 11
associated types, 271-274
associated values, 138-140
associativity, 29
attributes, 319
attributes inspector, 348, 374
Auto Layout, 326-328, 349-358
Automatic Reference Counting (ARC), 289

B
binary numbers, 25
Boolean variables, 21
break statements, 45, 56
bridging, 313
bridging headers, 382, 400
buttons
  adding to navigation bar, 389
  adding to view, 322, 348

C
catch statements, 246
Character type, 58
characters property, 58, 239
class keyword, 155, 162, 178
Class...has no initializers error, 194
classes
  about, 155
  computed properties, 178
  convenience initializers, 191, 197, 198
  creating, 155, 156
  default initializers, 191
  designated initializers, 191, 194-196
  failable initializers, 204
  inheritance (see inheritance)
  memory management, 200, 201
  memory usage, 289
  required subclass initializers, 199, 200
  root, 361
  stored properties, 178
  structs vs., 160, 209, 219
  using in Objective-C, 383, 384
closure expressions, 115
closures
  about, 113
  with lazy properties, 172
  map(_: ) method, 267
  reference cycles in, 295-298
  as reference types, 122, 123
  self in, 172
  shorthand argument names, 116
  tracking values with, 121, 122
  trailing syntax, 116
  type inference in, 115, 116
Cocoa
  Auto Layout, 326-328
  creating a new project, 315, 316
  error handling, 339
  interaction with Swift, 371
  loading documents, 339-341
  NSTextView, 318
  saving documents, 334-339
  speech synthesizer, 331
  view controllers, 338
  window controllers, 336, 338
Cocoa Touch Class, 360
collection types, 12
collections
  about, 75
  arrays (see arrays)
  dictionaries (see dictionaries)
  sets (see sets)
command-line tools, 4, 145
comments, 6
comparability, 301
Comparable protocol, 304-307
collectors (see comparison operators)
comparison operators
  about, 305
  overloading, 302-305
  table of, 20
condition expressions, 51
conditional statements
  else if, 24
  guard, 111, 249
  if-case, 45
  if/else, 19-21
  nested if’s, 23
  ternary operator, 22
  while let, 242
core, 8
constants
  declaring, 13
  as reference types, 213, 214
  as value types, 213, 214
  variables vs., 13
contains(_: ) method, 97
continue statements, 54, 55
control transfer statements
  break, 45, 56
  continue, 54, 55
  fallthrough, 38
Index

in loops, 54
controllers, 317
convenience keyword, 197
copies, shallow vs. deep, 216, 217
count property, 62, 78, 88
curried functions, 167
CustomStringConvertible protocol, 262, 283

D
data types, 11
debug area, 8, 148
decrementing, 30
default case, 35, 40, 41
deinit method, 290
deinitializers, 200, 201, 289
dictionaries
about, 87
adding items, 90
converting to arrays, 93
counting items, 88
declaring, 87, 88
immutable, 93
looping over, 92
modifying values, 89, 90
populating, 88
reading from, 89
removing items, 91
sets vs., 95
Dictionary type, 87
didSet, 176
division, 29
do/catch statements, 246
document outline, 320
document-based applications, 313
dot syntax, 58
Double type, 33

E
editor area, 148
else if statements, 24
empty variables, 14
encapsulation, 223
enumerations
about, 129
associated values, 138-140
comparing values, 130-132
creating, 129, 130
as ErrorTypes, 243
methods on, 135
nested, 170, 171
raw values, 132-134
recursive, 141-143
equality, 83, 84, 218, 270, 301
Equatable protocol, 270, 301-304
error domains, 339
error handling
assertions, 241
catching, 246, 247
in Cocoa, 339
ignoring, 252, 253
Swift philosophy of, 254, 255
throwing, 243, 244, 254
traps, 237
Errors thrown from here are not handled.
error, 252
effects, recoverable vs nonrecoverable, 237
ErrorType protocol, 243
Execution interrupted error, 30
exhaustiveness checks, 255
extension keyword, 258, 281
extensions
about, 257
adding functions, 263
adding initializers, 260, 261
adding nested types, 261, 262
on existing types, 257, 258
for grouping, 259
for protocol conformance, 259
on protocols, 279

F
fallthrough statement, 38
filter( _ : ) method, 125
first-class functions, 123
first-class objects, 117
Float type, 33
floating-point numbers, 33, 34
for case statements, 50, 51
for keyword, 47, 51, 52
for-in loops, 47-49, 92
frameworks
(see also modules)
free functions, 153
func keyword, 101
Index

function currying, 163-168
function types, 111, 112, 117, 118
functional programming, 123
functions, 8
   about, 101
   adding via an extension, 263
   as arguments, 118-120
   calling, 101
   curried, 163-165
   defining, 101
   generic, 267-269
   higher-order, 123-126
   modifying argument values, 106, 107
   nesting, 108
   overloading, 303
   parameters (see parameters)
   polymorphism, 277
   as return type, 117, 118, 163
   returning from, 111
   returning multiple values, 108-110
   returning optionals, 110
   returning values, 107
   scope, 108

G
GeneratorType protocol, 271-274
generics
   about, 265
   associated types, 271-274
   declaring, 266
   functions and methods, 267-269
   optionals, 276
   type constraints, 270
get keyword, 175
getters, 175, 182, 183
global functions, 153
guard statements, 111

H
hashability, 87
hexadecimal codes, in strings, 60
higher-order functions, 123-126

I
IB (Interface Builder) (see Xcode)
@IBAction, 331
@IBOutlet, 331
identity, 218
if-case statement, 45
if/else statements, 19-21, 23, 45, 46
immutability, 123, 215
import Foundation, 149
import keyword, 180
in-out parameters, 106, 107, 167
incrementing, 30
infinite loops, 55
inheritance
   about, 155-157
   class initializers and, 192, 193
   protocol, 233, 234
   init keyword, 185
   initialization, 14, 51, 185, 205
   initializer delegation, 189-191, 198
initializers
   adding via an extension, 260, 261
   automatic inheritance, 193
   class inheritance and, 192, 193
   convenience, 191, 194, 197, 198
   creating, 185, 369
   creating a set from an array, 96
   custom, 187-189
   default for classes, 191
   default for structs, 186, 187
   deinitialization, 200, 201, 289
   designated, 191, 194-196, 198, 200
   empty, 186
   failable, 201-204
   memberwise, 186-188, 190
   parameters, 206
   required, 199, 200
   inout keyword, 106
insert(_: method, 96
insert(_:atIndex:) function, 82
instance methods, 153, 154, 166
Int type, 12
   converting, 32
   declaring, 27, 28
   OS/X vs iOS, 26
   recommendation for, 32
   sized, 26
Int16 type, 26
Int32 type, 26
Int64 type, 26
Int8 type, 26
integer overflow error, 28
integers
  about, 25
  converting types, 32
  maximum and minimum values, 25-27
  operations on, 28-31
  overflow/underflow, 30, 31
  signed/unsigned, 26
Interface Builder (IB) (see Xcode)
  internal access, 181
  internal private(set) syntax, 182
intersect(_:) method, 98
interval matching, 44
iOS
  adding model to view controller, 365-367
  Auto Layout, 349-358
  creating a model, 360-365
  creating a new project, 346, 347
  image views, 396-400
  loading files, 369, 370
  navigation controller, 387-389
  saving files, 367-369
  table view, 354-357
isDisjointWith(_:) method, 99, 100
 iterators, 47, 49
K
  keys, 87, 89
 KeyType, 87
L
  last property, 216
  lazy keyword, 171
  lazy loading, 171
  let keyword, 13, 85, 93, 123, 170
  line breaks (in code), 152
  logical operators, 21
  loops
    about, 47
    over arrays, 80, 81
    control transfer statements in, 54-56
    over dictionaries, 92
    for, 51, 52
    for case statements in, 50, 51
    for-in, 47-49, 92
    infinite, 55
    repeat-while, 53
over sets, 96
while, 52, 53
while let, 242
M
  main.swift, 148, 149
  map(_:) method, 124, 231
  memory allocation, 289, 290
  memory leaks, 293
  memory management
    about, 289, 290
    deinitializers, 200, 201
    reference count, 289
    reference cycles, 293-298
    references, 290-293
  methods
    about, 113
    on enumerations, 135
    generic, 267-269
    mutating, 137, 154, 235, 236
    overloading, 303
    parameter names, 160
    type methods, 162
  Missing argument error, 192, 196
  Model-View-Controller (MVC) design, 317
  models, 228, 317, 360-367
  modules
    (see also frameworks)
    mutating keyword, 154, 163
    mutating methods, 137
    MVC (Model-View-Controller) design, 317
N
  navigator area, 148
  nested types, 170, 171, 261, 262
  next method on generators, 271
  nil coalescing operator, 71, 72
  nil value, 65, 66, 158, 203
  not operator (!), 21
  NSArray type, 368
  NSData class, 337
  NSDocument class, 338
  NSError class, 339
  NSFileManager class, 368
  NSObject class, 361, 383
  NSSpeechSynthesizer class, 331-333
  NSText property, 319

409
NStextView class, 318
**NSViewController** class, 318, 338
numbers
  floating-point, 33, 34

O
object library, 321
Objective-C
  adding Swift file, 382, 383
  sample project, 371-381
  using in Swift, 396-402
  using Swift in, 383, 384
optionals
  about, 65, 276
  accessing value of, 66, 71
  binding, 67-69, 111, 120, 158
  chaining, 70, 158, 159, 203
  declaring, 65
  force-unwrapping, 66
  implicitly unwrapped, 69
  modifying in place, 71
  nil coalescing operator, 71, 72
  returning, 110, 119, 201
  unwrapping multiple, 68
or operator (||), 21
Organization Identifier, 146
Organization Name, 146
outlets, 330, 331, 358-360, 392
overflow operators, 31
overloading, 303
override error, 179
override keyword, 179
overriding, 157-159, 179

P
parameters
  about, 102
  default values, 105, 106
  in-out, 106, 107, 167
  for initializers, 206
  in methods, 160
  names, 103, 104
  passing multiple, 102, 103
  variadic, 104
pattern matching, 44, 45, 139
placeholder types, 266, 270
playgrounds
  Assistant Editor view, 25
  code editor, 6
  creating, 4
  debug area, 8
  Quick Look, 78
  results sidebar, 6, 48
polymorphism, 277
precedence, 29
**print()**, 8
private access, 181
Product Name, 146
project navigator, 148
properties
  about, 169
  adding to a struct, 152
  computed, 169, 174, 178
  lazy stored, 171-173
  lazy, calculation of, 173
  optional, 155
  read-only, 169
  read/write, 169
  static, 179, 180
  stored, 169, 170, 178, 258
  type properties, 177-180
  property observers, 176, 177, 259
  protocol composition, 234, 235
  protocol conformance, 232, 233, 258, 259, 284
  protocol extensions
    about, 279
    creating, 281, 282
    default implementations, 283-285
    in the Swift standard library, 288
    naming conflicts, 286, 287
    where clauses, 282, 283
  protocol inheritance, 233, 234
  protocol keyword, 235
protocols
  about, 223, 229
  associated types, 271-274
  Comparable, 304-307
  CustomStringConvertible, 232
  defining, 229-232
  Equatable, 270, 301-304
  ErrorType, 243
  extending (see protocol extensions)
    as types, 230
public access, 181
pure functions, 123

410
Pyramid of Doom, 68

Q
quotation marks, 6

R
ranges, 39, 63
raw value enumerations, 132-134
recursive enumerations, 141-143
reduce(_:combine:), 126
refactoring, 23
reference count, 289
reference cycles, 293-298
reference semantics, 209-211
reference types
about, 122, 123
constants as, 213, 214
in value types, 214, 215
references, 209, 211, 290-295
relationship segues, 388
removeAtIndex(_:) function, 78
removeValueForKey(_:) function, 91
repeat-while loops, 53
'required' initializer error, 199
required keyword, 199
return keyword, omitting in closures, 116
return values
about, 107
functions as, 117, 118
multiple, 108-110
optional, 110, 119
root classes, 361
run program, 149

S
scenes, 321, 378
scope, 108
segues, 388, 390, 391, 393, 394
self, 135-138, 172, 236
SequenceType protocol, 272, 282
set keyword, 175
Set type, 95
sets
about, 95
adding items, 96
checking for an item, 97
combining, 97, 98
declaring, 95-97
disjoint, 99, 100
intersections, 98
looping over, 96
setters, 175, 176, 182, 183
sleep() function, 54
sort(_::) method, 113-117
source files, 180
specialization
of generics, 267
speech, synthesizing, 331-333
startIndex property, 62
static keyword, 177, 178
static property, 179, 180
storyboards, 320, 374
root view controllers, 388
string interpolation, 15, 40, 92, 110
String type, 11, 57
String variables, 6, 7
String.Index type, 62
strings
accessing characters in, 62, 63
canonical equivalence, 61, 62
counting characters, 62
creating, 57, 58
inserting variable value in, 15
mutable, 57
Unicode scalars, 59-61
strong typing, 123
struct keyword, 152
structs
about, 150
adding properties, 152
changing properties of, 154
classes vs., 160, 209, 219
creating, 152
custom initializers, 187-189
default initializers, 186, 187
instance methods, 153, 154
subclasses
about, 156
creating, 156, 157, 386
overriding, 157-159, 178
requiring initializers on, 199, 200
subscript syntax ([ ]), 62
subscripting, 79, 80, 89
super keyword, 157
superclasses, 156, 195
Index

switch cases
  about, 35, 36
  associated values and, 139
  basic example, 36-38
  if-case, 45
  if/else vs., 45, 46
  pattern matching in, 44, 45
  ranges, 39
tuples, 42-44
  where clauses, 41, 42
switch statements, 130-132

T
target-action pairs, 329, 358-360
text fields, 349
text view
  capturing entered text, 330, 331
  empty, 332
  loading data into, 339-341
  managing with ViewController, 342, 343
  throws keyword, 242-244, 249
  trailing closure syntax, 116
treaps, 31, 237
trees, 141-143
try keyword, 246, 250, 252-254
tuples, 42-44, 108-110
type annotation, 12
Type annotation missing in pattern error, 187
type casting, 337
type checking, 11
type constraints, 270, 272, 274, 275
type extensions (see extensions)
type inference, 11, 51, 76, 115, 116, 130, 272, 273
type methods, 162
  type properties, 177-180
  type...does not conform to protocol... error, 236
  typealias keyword, 257, 258, 271

U
  UIImageView type, 396-400
  UIKit, 386
  UINavigationController type, 387
  UInt type, 27
  UITableView type, 354-357, 363
  underflowing, 31
  Unicode, 59
  Unicode scalars
    about, 59-61
    canonical equivalence, 61, 62
    combining, 60
    extended grapheme clusters, 60
    unicodeScalars property, 60
  union(_: ) method, 97, 98
  unwind segues, 393, 394
  unwrapping
    forced, 66, 204
    implied, 69
    multiple optionals, 68
  updateValue(_: forKey: ) function, 89, 90
  upper case, converting to, 70
  Use of 'self' in delegating initializer
  before self.init is called error, 197
  utilities area, 148, 321

V
  value binding, 40, 41
  Value of optional type...not unwrapped... error, 203
  value semantics, 207-209
  value types
    about, 209
    constants as, 213, 214
    memory management, 200
    in reference types, 214, 215
    self and, 137
  var keyword, 6
  variable types, 7, 11, 21
  variables, 6
    assigning function types to, 112
    assigning values, 7
    constants vs., 13
    declaring, 6, 12
    initializing, 14
    inserting into a string, 15
  variadic parameters, 104
  view controllers, 321, 338, 375-377, 393
  views
    about, 317
    adding buttons, 322, 348
    adding text field, 349
    creating, 320, 321, 348, 349

412
Index

... dismissing, 394
image, 396-400
layout, 349-358
table, 354-357, 362-367, 378
text, 323-325

W
where clauses
  on generic bounds, 274, 275
  in protocol extensions, 282, 283
  on if cases, 46
  on switch cases, 41, 42
where clauses on optional binding, 69
while let loops, 242
while loops, 52, 53
willSet, 176, 259
writeToURL method, 368

X
Xcode
  adding a file, 150
  adding a text view, 323-325
  application window, 148
  connecting outlets, 330, 331
  creating a model, 360-365
  creating a new project, 145, 315, 316, 346, 347
  creating views, 320, 321, 348, 349
  Custom Class, 378
  document outline, 320
  importing header files, 384, 400
  Interface Builder (IB), 319
  issue navigator, 189
  object library, 321
  playgrounds (see playgrounds)
  target-action pairs, 329, 358-360
  toolbar, 149
  utilities area, 321
  view element declaration, 288
  view layout, 326-328, 349-358
  Xcode documentation, 85