Swift 3 for Absolute Beginners

Third Edition

Gary Bennett
Brad Lees

Apress®

Swift 3 for Absolute Beginners

Third Edition

Gary Bennett
Brad Lees

Apress®

Swift 3 for Absolute Beginners

Gary Bennett
Scottsdale, Arizona, USA

Brad Lees
Phoenix, Arizona, USA

ISBN-13 (pbk): 978-1-4842-2330-7
ISBN-13 (electronic): 978-1-4842-2331-4
DOI 10.1007/978-1-4842-2331-4

Library of Congress Control Number: 2016962063

Managing Director: Welmoed Spahr
Lead Editor: Aaron Black
Technical Reviewer: Stefan Kaczmarek
Editorial Board: Steve Anglin, Pramila Balen, Louise Corrigan, James DeWolf, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing
Coordinating Editor: Jessica Vakili
Copy Editor: Ann Dickson
Compositor: SPi Global
Indexer: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springer.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Printed on acid-free paper

Contents at a Glance

Contents

About the Authors

Gary Bennett is president of xcelMe.com. xcelMe teaches iPhone/iPad programming courses online. Gary has taught hundreds of students how to develop iPhone/iPad apps. He has created several very popular apps himself, and his students have some of the best-selling apps on the iTunes App Store. Gary also worked for 25 years in the technology and defense industries. He served 10 years in the U.S. Navy as a nuclear engineer aboard two nuclear submarines. After leaving the Navy, Gary worked for several companies as a software developer, CIO, and president. As CIO, he helped take VistaCare public in 2002. Gary also co-authored *iPhone Cool Projects* for Apress. Gary lives in Scottsdale, Arizona, with his wife Stefanie and their four children.

Brad Lees has more than 12 years' experience in application development and server management. He has specialized in creating and initiating software programs in real-estate development systems and financial institutions. His career has been highlighted by his positions as information systems manager at The Lyle Anderson Company; product development manager for Smarsh; vice president of application development for iNation; and IT manager at The Orcutt/Winslow Partnership, the largest architectural firm in Arizona. A graduate of Arizona State University, Brad and his wife Natalie reside in Phoenix with their five children.

About the Technical Reviewer

Stefan Kaczmarek has over 15 years of software development experience specializing in mobile applications, large-scale software systems, project management, network protocols, encryption algorithms, and audio/video codecs. As chief software architect and co-founder of SKJM, LLC, Stefan developed a number of successful mobile applications including iCam (which has been featured on CNN, *Good Morning America*, *The Today Show*, and the "Dog Lover" iPhone 3GS television commercial) and iSpy Cameras (which held the #1 Paid iPhone App ranking in a number of countries around the world including the UK, Ireland, Italy, Sweden, and South Korea). Stefan resides in Phoenix, Arizona, with his wife Veronica and their two children.

Introduction

Over the last seven years, we've heard the following comments countless times:

- "I've never programmed before, but I have a great idea for an iPhone/iPad/AppleTV app."
- "Can I really learn to program the iPhone or iPad?"

To the latter we answer, "Yes, but you have to believe you can." Only you are going to tell yourself you can't do it.

For the Newbie

This book assumes you may have never programmed before. The book is also written for someone who may have never programmed before using object-oriented programming (OOP) languages. There are many Swift books out there, but all of these books assume you have programmed before and know OOP and computer logic. We wanted to write a book that takes readers from knowing little or nothing about computer programming and logic to being able to program in Swift. After all, Swift is a native programming language for the iPhone, iPad, and Mac.

Over the last seven years, we have taught thousands of students at xcelMe.com to be iPhone/iPad (iOS) developers. Many of our students have developed some of the most successful iOS apps in their category in the iTunes App Store. We have incorporated what we have learned in our first two courses, "Introduction to Object-Oriented Programming" and "Logic and Swift for iPhone/iPad Developers," into this book.

For the More Experienced

Many developers who programmed years ago or programmed in a non-OOP language need a background in OOP and Logic before they dive Swift. This book is for you. We gently walk you through OOP and how it is used in iOS development to help make you a successful iOS developer.

How This Book Is Organized

You'll notice that we are all about successes in this book. We introduce the OOP and Logic concepts in Playgound and then move those concepts to Xcode and Swift. Many students are visual learners or they learn by doing. We use both techniques. We'll walk you through topics and concepts with visual examples and then take you through step-by-step examples while reinforcing the concepts.

We often repeat topics in different chapters to reinforce what you have learned and apply these skills in new ways. This enables new programmers to reapply development skills and feel a sense of accomplishment as they progress. Don't worry if you feel you haven't mastered a topic. Keep moving forward!

The Formula for Success

Learning to program is an interactive process between your program and you. Just like learning to play an instrument, you have to practice. You must work through the examples and exercises in this book. Understanding the concept doesn't mean you know how to apply it and use it.

You will learn a lot from this book. You will learn a lot from working through the exercises in this book. However, you will really learn when you debug your programs. Spending time walking through your code and trying to find out why it is not working the way you want is an unparalleled learning process. The downside of debugging is a new developer can find it especially frustrating. If you have never wanted to throw your computer out the window, you will. You will question why you are doing this and whether you are smart enough to solve the problem. Programming is very humbling, even for the most experienced developer.

Like a musician, the more you practice the better you get. By practicing, we mean programming! You can do some amazing things as a programmer. The world is your oyster. Seeing your app in the iTunes App Store is one of the most satisfying accomplishments. However, there is a price, and that price is time spent coding and learning.

Having taught many students to become iOS developers, we have put together a formula for what makes students successful. Here is our formula for success:

- Believe you can do it. You'll be the only one who says you can't do this, so don't tell yourself that.
- Work through all the examples and exercises in this book.
- Code, code, and keeping coding. The more you code, the better you'll get.
- Be patient with yourself. If you were fortunate enough to have been a 4.0 student who could memorize material just by reading it, don't expect your memorization skills to translate to easy success in Swift coding. The only way you are going to learn is to spend time coding.
- You learn by reading this book. You really learn by debugging your code.
- Use the free `xcelMe.com` webinars and YouTube videos explained at the end of this introduction.
- Don't give up!

The Development Technology Stack

We will walk you through the process of understanding the development process for your iOS apps and what technology you need. However, briefly looking at all the technology pieces together is helpful. These are the key iOS development technology pieces you will need to know in order to build a successful app and get it on the app store:

- Apple's Developer Website
- App Telemetry
- App Analytices
- iPhone Swift SDK

- Swift
- Object-Oriented Programming and Logic
- Xcode IDE
- Debugging
- Performance Tuning

We know this is a lot of technology. Don't worry—we will go through it and will be comfortable using it.

Required Software, Materials, and Equipment

One of the great things about developing iOS apps is just about everything you need is free to develop your app.

- Xcode
- Swift
- macOS Sierra10.12.1 or higher
- Xcode Integrated Developers Environment
- iOS SDK
- iPhone and iPad Simulator

All you need to get started is a Mac and knowledge of where to download everything, which we will cover.

Operating System and IDE

When developing iOS apps, you have to use Xcode and the macOS. You can download both of these for free from the Mac App Store (see Figure 1.)

Figure 1. *The Mac App Store*

Software Development Kits

You will need to register as a developer. You can do this for free at `http://developer.apple.com/ios` (see Figure 2).

Figure 2. *Apple's Developer website*

When you are ready to upload your app to the iTunes App Store, you will need to pay $99/year in order to obtain access.

Figure 2 Apple's Developer Website (editor, caption not sure why I can't apply that style)

Dual Monitors [editor not sure why this "Strong" format is doing this]

We recommend developers have a second monitor connected to their computer. It is great to step through your code and watch your output window and iPad simulator at the same time on dual independent monitors.

Apple hardware makes this easy. Just plug your second monitor into the port of any Mac, with the correct adapter of course, and you have two monitors working independently of one another. (See Figure 3.) Note that dual monitors are not required. You will just have to organize your open windows to fit on your screen if you don't have two monitors.

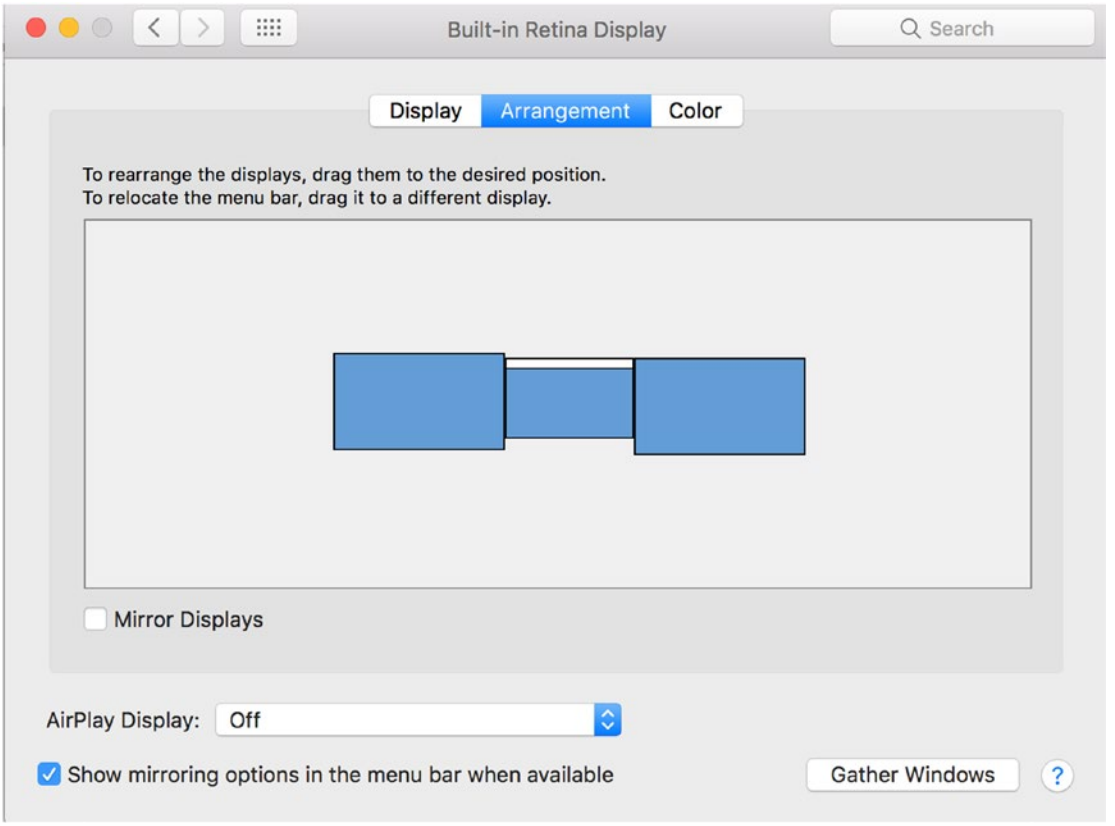

Figure 3. *Configuring a second monitor*

Nearly every week, we have live webinars and discuss a topic from the book or a timely item of interest. These webinars are free, and you can register for them at `www.xcelme.com/latest-videos/`. See Figure 4.

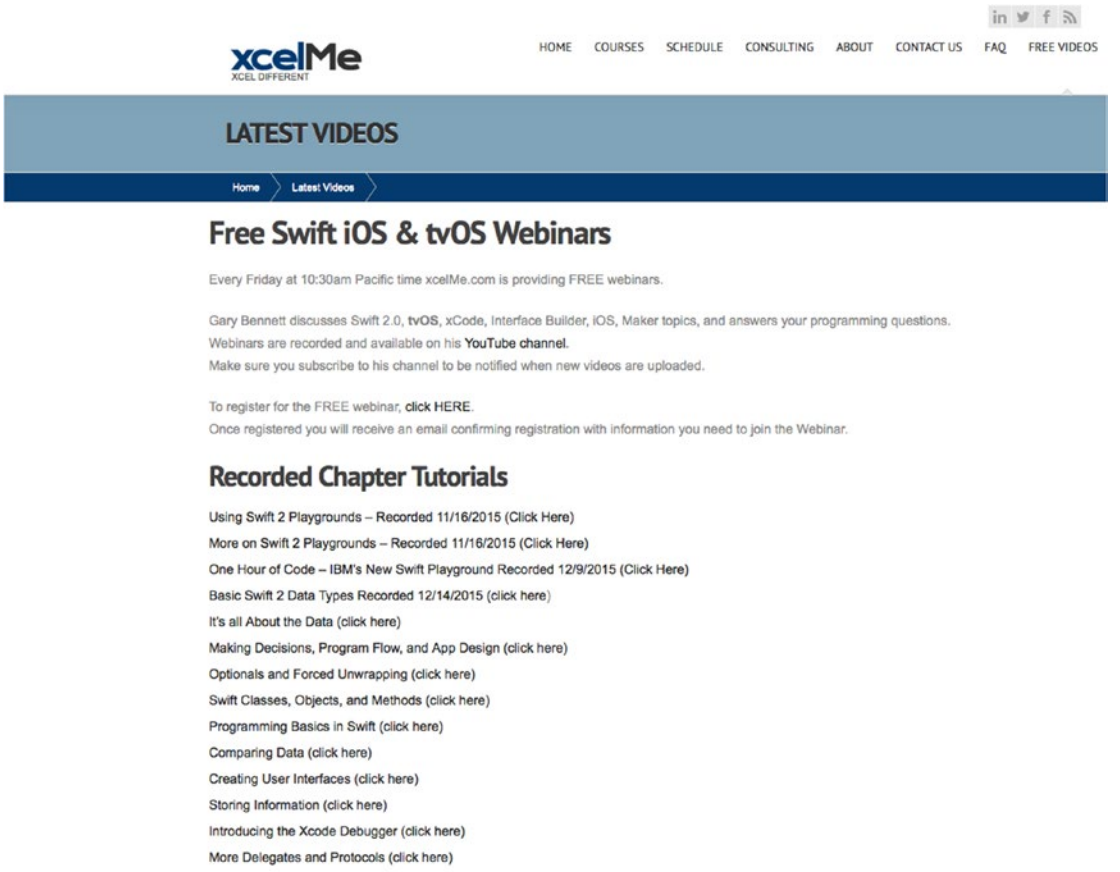

Figure 4. *Register for free webinars at* `www.xcelme.com/latest-videos/`

At the end of the webinars, we do a Q&A. You can ask a question on the topic discussed or any topic in the book.

Additionally, all the webinars are recorded and available on YouTube. Make sure you subscribe to the YouTube channel so you are notified when new recordings are uploaded.

Free Book Forum

We have developed an online forum for this book at `http://forum.xcelme.com`, where you can ask questions while you are learning Swift and get answers from the authors. You will also find answers to the exercises and additional exercises to help you learn. (See Figure 5.)

Figure 5. *Reader forum for accessing answer to exercise and posting questions for authors*

CHAPTER 1

Becoming a Great iOS Developer

Now that you're ready to become a software developer and have read the introduction of this book, you need to become familiar with several key concepts. Your computer program will do exactly what you tell it to do—no more and no less. It will follow the programming rules that were defined by the operating system and the Swift programming language. Your program doesn't care if you are having a bad day or how many times you ask it to perform something. Often, what you think you've told your program to do and what it actually does are two different things.

Key to Success If you haven't already, take a few minutes to read the introduction of this book. The introduction shows you where to go to access the free webinars, forums, and YouTube videos that go with each chapter. Also, you'll better understand why this book uses the Swift playground programming environment and how to be successful in developing your iOS apps.

Depending on your background, working with something absolutely black and white may be frustrating. Many times, programming students have lamented, "That's not what I wanted it to do!" As you begin to gain experience and confidence in programming, you'll begin to think like a programmer. You will understand software design and logic, and you will experience having your programs perform exactly as you want, and you will enjoy the satisfaction associated with this.

1.1 Thinking Like a Developer

Software development involves writing a computer program and then having a computer execute that program. A *computer program* is the set of instructions that you want the computer to perform. Before beginning to write a computer program, it is helpful to list the steps that you want your program to perform in the order you want them accomplished. This step-by-step process is called an *algorithm*.

If you want to write a computer program to toast a piece of bread, you would first write an algorithm. The algorithm might look something like this:

1. Take the bread out of the bag.
2. Place a slice of bread in the toaster.
3. Press the "toast" button.
4. Wait for the toast to pop up.
5. Remove the toast from the toaster.

G. Bennett and B. Lees, *Swift 3 for Absolute Beginners*, DOI 10.1007/978-1-4842-2331-4_1

At first glance, this algorithm seems to solve the problem. However, the algorithm leaves out many details and makes many assumptions. Here are some examples:

- What kind of toast does the user want? Does the user want white bread, wheat bread, or some other kind of bread?
- How does the user want the bread toasted? Light or dark?
- What does the user want on the bread after it is toasted: butter, margarine, honey, or strawberry jam?
- Does this algorithm work for all users in their cultures and languages? Some cultures may have another word for toast or not know what toast is.

Now, you might be thinking this is getting too detailed for making a simple toast program. Over the years, software development has gained a reputation of taking too long, costing too much, and not being what the user wants. This reputation came to be because computer programmers often start writing their programs before they have actually thought through their algorithms.

The key ingredients to making successful applications are *design requirements*. Design requirements can be formal and detailed or simple like a list on a piece of paper. Design requirements are important because they help the developer flesh out what the application should and should not do when complete. Design requirements should not be completed in a programmer's vacuum, but should be produced as the result of collaboration between developers, users, and customers.

Another key ingredient to your successful app is the ***user interface*** (UI) design. Apple recommends you spend more than 50 percent of the entire development process focusing on the UI design. The design can be done using simple pencil and paper or using Xcode's storyboard feature to lay out your screen elements. Many software developers start with the UI design, and after laying out all the screen elements and having many users look at paper mock-ups, they write the design requirements from their screen layouts.

■ **Note** If you take anything away from this chapter, let it be the importance of considering design requirements and user interface design before starting software development. This is the most effective (and least expensive) use of time in the software development cycle. Using a pencil and eraser is a lot easier and faster than making changes to code because you didn't have others look at the designs before starting to program.

After you have done your best to flesh out all the design requirements, laid out all the user interface screens, and had the clients or potential customers look at your design and give you feedback, you can begin coding. Once coding begins, design requirements and user interface screens can change, but the changes are typically minor and are easily accommodated by the development process. See Figures 1-1 and 1-2.

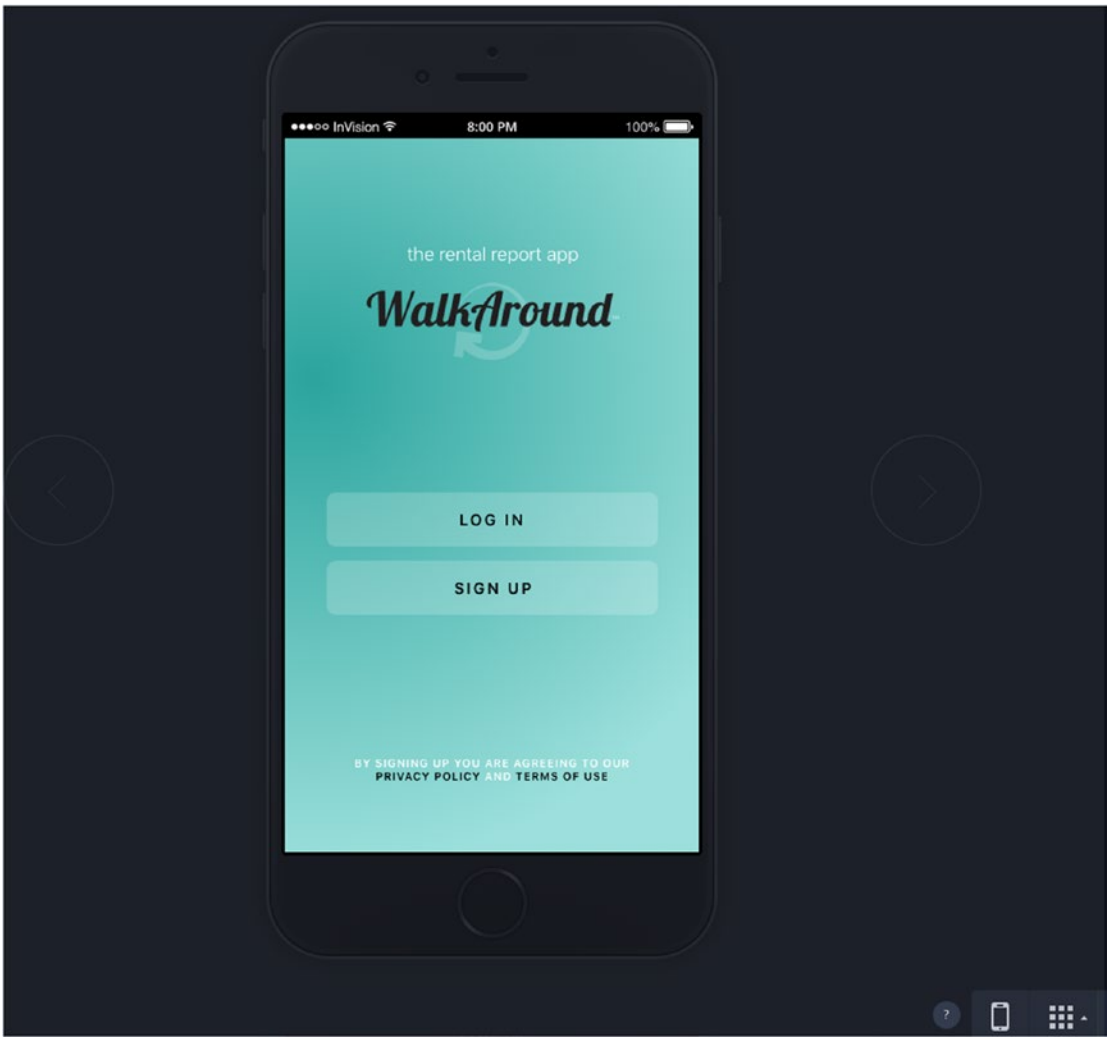

Figure 1-1. *This is a UI mock-up of the Log In screen for an iPhone mobile rental report app before development begins. This UI design mock-up was completed using InVision*

Figure 1-2. *This is the completed iPhone rental report app. This app is called WalkAround*

Figure 1-1 shows a mock-up of a rental report app screen prior to development. Developing mock-up screens along with design requirements forces developers to think through many of the application's usability issues before coding begins. This enables the application development time to be shortened and makes for a better user experience and better reviews on the App Store. Figure 1-2 shows how the view for the rental report app appears when completed. Notice how mock-up tools enable you to model the app to the real thing.

Completing the Development Cycle

Now that you have the design requirements and user interface designs and have written your program, what's next? After programming, you need to make sure your program matches the design requirements and user interface design and ensure that there are no errors. In programming vernacular, errors are called *bugs*. Bugs are undesired results of your programming and must be fixed before the app is released to the App Store. The process of finding bugs in programs and making sure the program meets the design requirements is called ***testing***. Typically, someone who is experienced in software testing methodology and who didn't write the app performs this testing. Software testing is commonly referred to as ***quality assurance*** (QA).

■ **Note** When an application is ready to be submitted to the App Store, Xcode gives the file an `.app` or `.ipa` extension, for example, `appName.app`. That is why iPhone, iPad, and Mac applications are called ***apps***. This book uses *program*, *application*, and *app* to mean the same thing.

During the testing phase, the developer will need to work with the QA staff to determine why the application is not working as designed. The process is called *debugging*. It requires the developer to step through the program to find out why the application is not working as designed. Figure 1-3 shows the complete software development cycle.

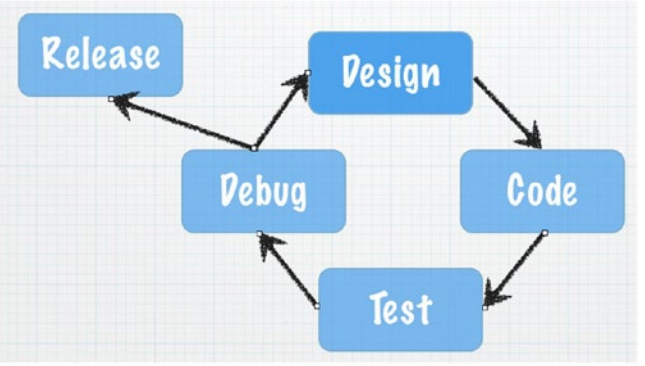

Figure 1-3. *The typical software development cycle*

Frequently during testing and debugging, changes to the requirements (design) must occur to make the application more usable for the customers. After the design requirements and user interface changes are made, the process starts again.

At some point, the application that everyone has been working so hard on must be shipped to the App Store. Many considerations are taken into account as to when in the cycle this happens:

- Cost of development
- Budget
- Stability of the application
- Return on investment

There is always the give-and-take between developers and management. Developers want the app to be perfect, and management wants to start realizing revenue from the investment as soon as possible. If the release date were left up to the developers, the app would likely never ship to the App Store. Developers would continue to tweak the app forever, making it faster, more efficient, and more usable. At some point, however, the code needs to be pried from the developers' hands and uploaded to the App Store so it can do what it was meant to do.

Introducing Object-Oriented Programming

As discussed in detail in the introduction, playgrounds enable you to focus on ***object-oriented programming*** **(OOP)** without having to cover all the Swift programming syntax and complex Xcode development environment in one big step. Instead, you can focus on learning the basic principles of OOP and using those principles quickly to write your first programs.

For decades, developers have been trying to figure out a better way to develop code that is reusable, manageable, and easily maintained over the life of a project. OOP was designed to help achieve code reuse and maintainability while reducing the cost of software development.

OOP can be viewed as a collection of objects in a program. Actions are performed on these objects to accomplish the design requirements.

An ***object*** is anything that can be acted on. For example, an airplane, person, or screen/view on the iPad can all be objects. You may want to act on the plane by making the plane bank. You may want the person to walk or to change the color of the screen of an app on the iPad.

Playgrounds execute your code as you complete each line, such as the one shown in Figure 1-4. When you run your playground applications, the user can apply actions to the objects in your application. Xcode is an ***integrated development environment*** (IDE) that enables you to run your application from within your programming environment. You can test your applications on your computer first before running them on your iOS devices by running the apps in Xcode's simulator, as shown in Figure 1-5.

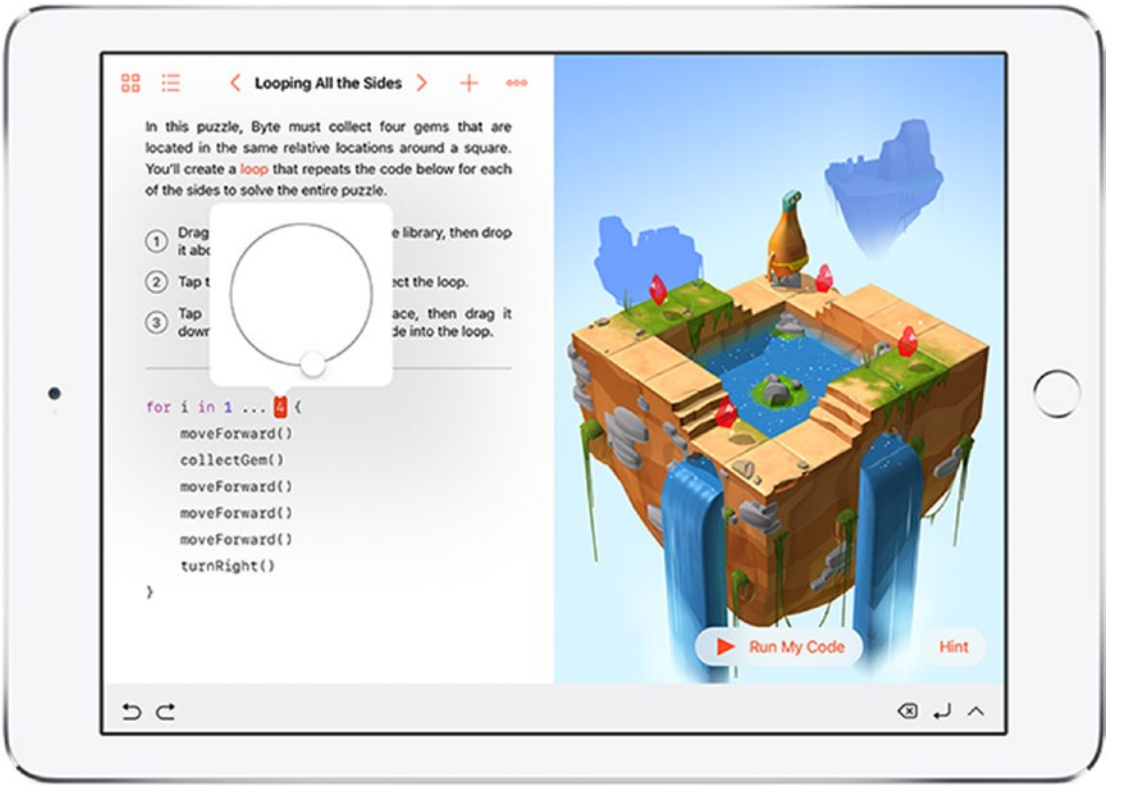

Figure 1-4. *There are multiple objects in this playground view*

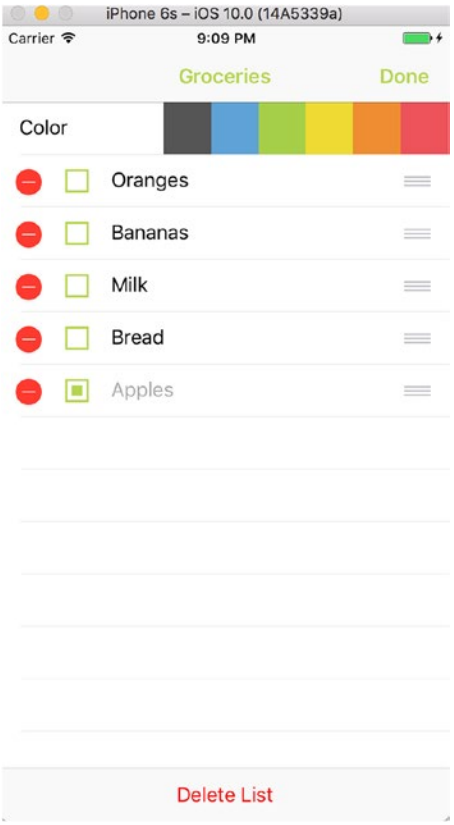

Figure 1-5. *This sample iPhone app contains a table object to organize a list of groceries. Actions such as "rotate left" or "user did select row 3" can be applied to this object.*

Actions that are performed on objects are called ***methods***. Methods manipulate objects to accomplish what you want your app to do. For example, for a `jet` object, you might have the following methods:

```
goUp
goDown
bankLeft
turnOnAfterburners
lowerLandingGear
```

The `table` object in Figure 1-5 is actually called `UITableView` when you use it in a program, and it could have the following methods:

```
numberOfRowsInSection
cellForRowAtIndexPath
canEditRowAtIndexPath
commitEditingStyle
didSelectRowAtIndexPath
```

Most objects have data that describes those objects. This data is defined as *properties***.** Each property describes the associated object in a specific way. For example, the `jet` object's properties might be as follows:

```
altitude = 10,000 feet
heading = North
speed = 500 knots
pitch = 10 degrees
yaw = 20 degrees
latitude = 33.575776
longitude = -111.875766
```

For the `UITableView` object in Figure 1-5, the following might be the properties:

```
whiteGroundColor = Red
selectedRow = 3
animateView = No
```

An object's properties can be changed at any time when your program is running, when the user interacts with the app, or when the programmer designs the app to accomplish the design requirements. The values stored in the properties of an object at a specific time are collectively called the ***state of an object***.

State is an important concept in computer programming. When teaching students about state, we ask them to go over to a window and find an airplane in the sky. We then ask them to snap their fingers and make up some of the values that the plane's properties might have at that specific time. Those values might be as follows:

```
altitude = 10,000 feet
latitude = 33.575776
longitude = -111.875766
```

Those values represent the *state* of the object at the specific time that they snapped their fingers.

After waiting a couple minutes, we ask the students to find that same plane, snap their fingers again, and record the plane's possible state at that specific point in time.

The values of the properties might then be something like the following:

```
altitude = 10,500 feet
latitude = 33.575665
longitude = -111.875777
```

Notice how the state of the object changes over time.

Working with the Playground Interface

Playgrounds offer a great approach to using the concepts just discussed without all the complexity of learning Xcode and the Swift language at the same time. It takes only a few minutes to familiarize yourself with the playground interface and begin writing a program.

Technically speaking, the playground interface is not a true IDE like you will be using to write your iOS apps, but it is pretty close and much easier to learn in. A true IDE combines code development, user interface layout, debugging tools, documentation, and simulator/console launching for a single application; see Figure 1-6. However, playgrounds offer a similar look, feel, and features to the Xcode IDE you develop apps with.

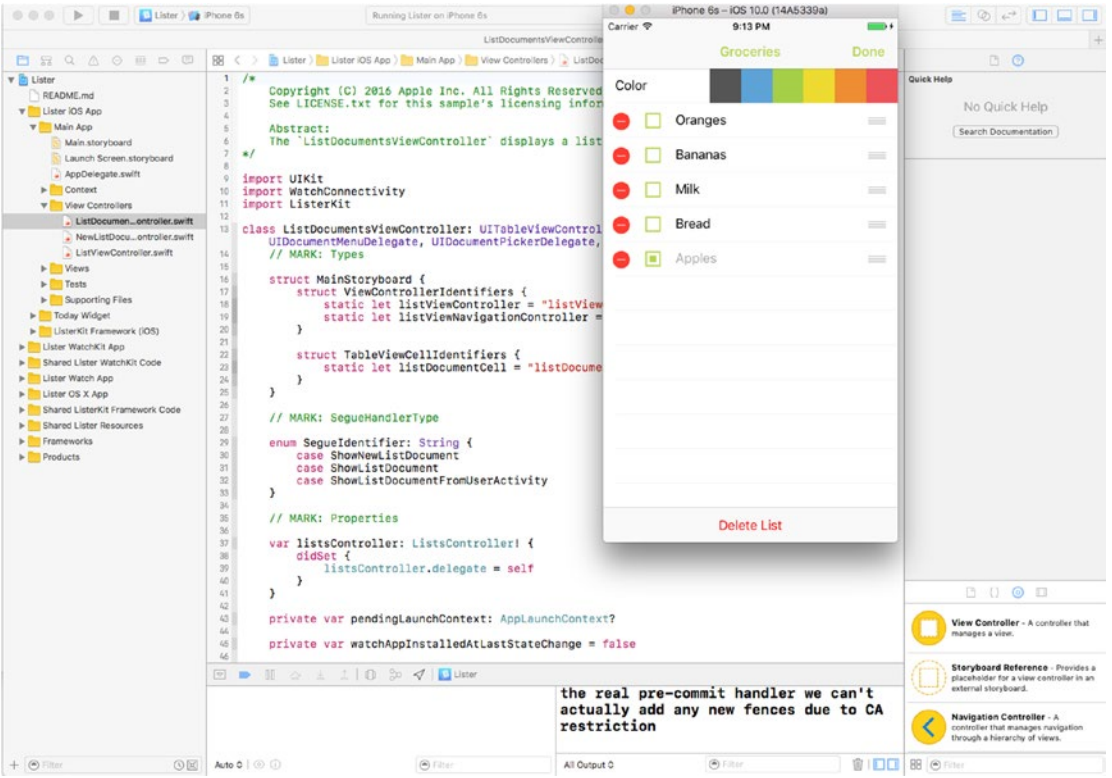

Figure 1-6. *The Xcode IDE with the iPhone simulator*

In the next chapter, you will go through the playground interface and write your first program.

Summary

Congratulations, you have finished the first chapter of this book. It is important that you have an understanding of the following terms because they will be reinforced throughout this book:

- Computer program
- Algorithm
- Design requirements
- User interface