The Expert's Voice in Swift

Swift Quick Syntax Reference

Matthew Campbell

Apress®

Contents at a Glance

Introduction

The expressions of the WWDC 2014 audience quickly changed from excitement and enthusiasm to looks of shock, horror, and awe. At this WWDC, after a succession of ever-surprising announcements, Apple ended the conference by announcing a completely new programming language designed entirely for Mac and iOS applications. This programming language is named Swift, which is what I have written about in this book.

The audience members' looks of shock and horror were understandable in context. Most of the developers at that conference had spent the past six years mastering the previous, relatively obscure programming language used to develop apps called Objective-C. The people sitting in those seats were world-class experts in a programming language that was just declared dead right in front of them.

What many of these developers probably had long since forgotten was just how difficult it is for most people to use Objective-C at first. Objective-C is also missing many features that other programmers take for granted such as tuples and generics. This is likely why that over the summer of 2014 many developers would become quite enthusiastic about adopting Swift in their projects.

It didn't take long for me to get on board with Swift. My initial reaction was relief that Apple decided to clean up the syntax, remove the clutter associated with Objective-C, and eject nonmainstream notions like messaging objects. I could tell immediately that the students I teach would take to Swift way more quickly than Objective-C.

This is one of the reasons I was so excited to write this book with Apress. Swift is absolutely the programming language that will take iOS and Mac into the future. Swift is a dramatic improvement to the application ecosystem. If you were turned off from making applications before because of Objective-C, now is the time to give making your app another go.

This book is written for programmers who want to get up to speed quickly in Swift. I made an effort to keep chapter headings specific, simple, and clear so you can go right to the area that you need to focus on. Chapters are short and focus on the syntax, but concepts are briefly illustrated at times when the information is crucial to the programming concepts that I'm presenting.

Since Swift is so new, I didn't make many assumptions about your technical background, so anyone with a general understanding of programming will benefit from this book. If you know what loops, functions, and objects are, you can follow the content here. Any niche or advanced programming constructs will be explained so you can follow along.

Good luck with your app! I hope that this book will help you appreciate Swift and see how this new language will make your life and your app much better.

Chapter 1

Hello World

I will start our conversation about Swift with the venerable Hello World program. However, you need to get some things in place before I can do that. Most importantly, you need a Mac app that will help you write and test Swift code. This Mac app is called Xcode.

Xcode

Xcode is a free app that you can download from the Apple App Store. Xcode gives you all the tools that you need to build applications for the Mac and iOS devices. These tools include a code editor, debugging tools, and everything else you need to turn your Swift code into an app.

Note Xcode requires a Mac with OS X 10.9.3 or OS X 10.10. You cannot install Xcode on a Windows- or Linux-based computer.

Install Xcode

To install Xcode, go to the Mac App Store by selecting your Mac's menu bar, clicking the Apple symbol, and then clicking App Store. Use the App Store search feature to locate Xcode by typing the word *Xcode* into the text box next to the hourglass. Press Return to search for Xcode. You will be presented with a list of apps, and Xcode should be the first app in the list. Install Xcode by clicking the button with the word *free* next to the Xcode icon. The word *free* changes to *installed* once it's ready to go, as shown in Figure 1-1.

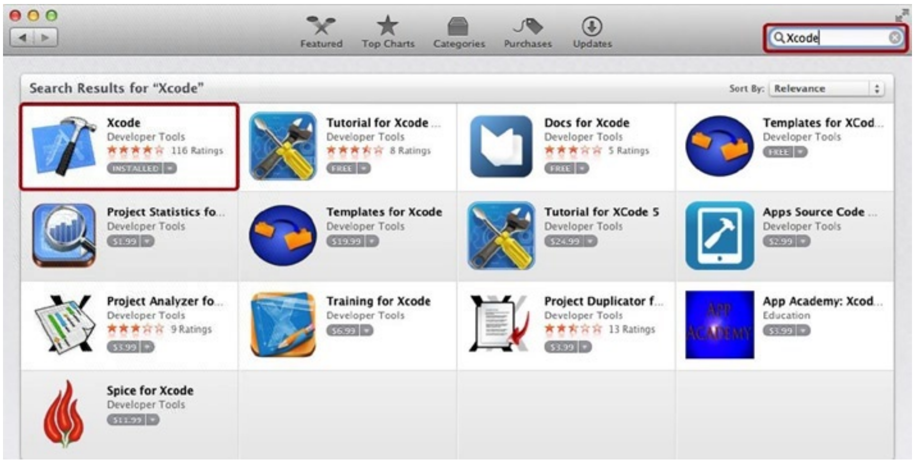


***Figure 1-1.** Downloading Xcode from the App Store*

Note Xcode version 6 is required to do Swift programming. By the time this book is released, Xcode 6 should be available in the Apple App Store, and you should be able to get it by following the previous instructions. However, at the time of this writing, Xcode 6 is still in beta and available only to registered Apple developers who can download it from the Apple developer web site at `http://developer.apple.com`.

Create a New Playground

Playgrounds are a workspace that you use to quickly prototype Swift code. The examples in this book will assume that you are using playgrounds to follow along. You use Xcode to make a playground.

Open Xcode by going to your `Applications` folder and clicking the Xcode app. You will be presented with a welcome screen. Click the text "Get started with a playground" to build your playground (see Figure 1-2).

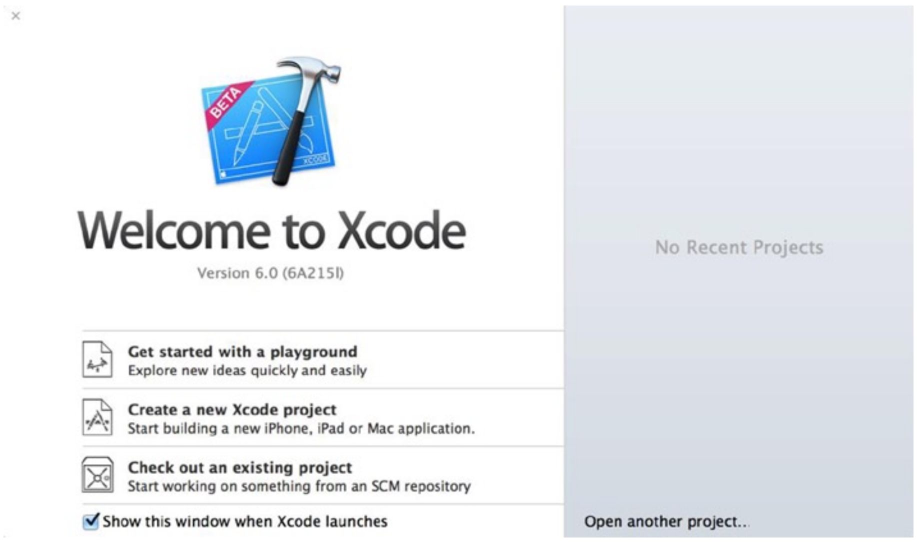

Welcome to Xcode
Version 6.0 (6A215l)
Get started with a playground
Explore new ideas quickly and easily
Create a new Xcode project
Start building a new iPhone, iPad or Mac application.
Check out an existing project
Start working on something from an SCM repository
Show this window when Xcode launches
No Recent Projects
Open another project...

Figure 1-2. Xcode welcome screen

You will be presented with a Save As screen, as shown in Figure 1-3. Use this screen to choose a name and location for your Swift playground.

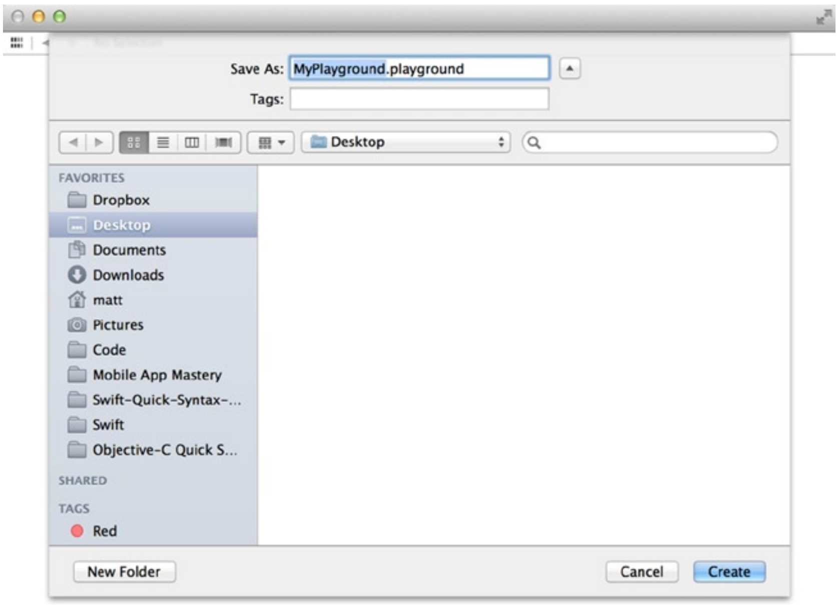
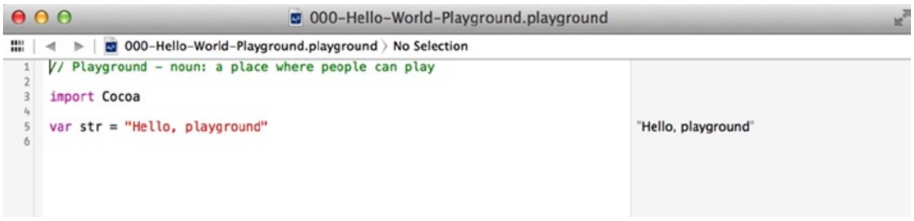

Save As: MyPlayground.playground
Tags:

Figure 1-3. Playground Save As screen

Once you choose your playground's name and folder location, Xcode will present a code editor with some boilerplate code already filled in for you (see Figure 1-4).


000-Hello-World-Playground.playground
000-Hello-World-Playground.playground 〉 No Selection

```
// Playground - noun: a place where people can play

import Cocoa

var str = "Hello, playground"
```

```
"Hello, playground"
```

***Figure 1-4.** Playground boilerplate*

Your first playground has been created, and you already have a sort of Hello World program coded for you. Well, not exactly. Your code says "Hello, playground," as you can see in Listing 1-1.

Listing 1-1. Hello Playground

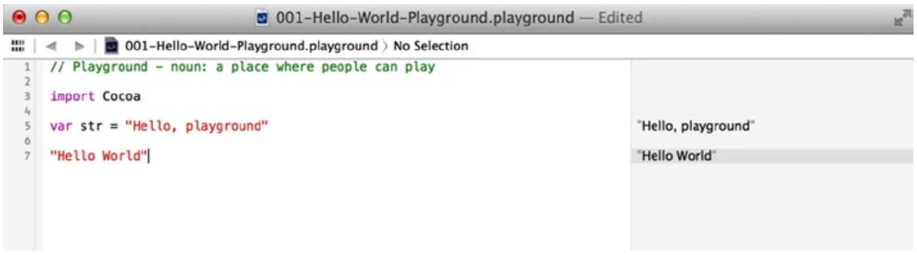
```
// Playground - noun: a place where people can play
import Cocoa
var str = "Hello, playground"
```

You use a playground by typing in code in the left area of the code editor. You can immediately see results appear on the right in the playground. To create a Hello World program, all you need to do is type in the phrase `"Hello World"` (including the quotes) into the playground code editor (see Listing 1-2).

Listing 1-2. Hello World

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

When you type in `"Hello World"`, you will immediately see the output "Hello World" appear on the right. See Figure 1-5 as a reference.


001-Hello-World-Playground.playground — Edited
001-Hello-World-Playground.playground 〉 No Selection

```
// Playground - noun: a place where people can play

import Cocoa

var str = "Hello, playground"

"Hello World"
```

```
"Hello, playground"
"Hello World"
```

Figure 1-5. "Hello World" output

Chapter 2

Declaring Constants and Variables

While you can use values like the string "`Hello World`" from the previous chapter or a number like 3.14 directly in code, usually you assign values like these to either a variable or a constant. You can give values a convenient name using variables and constants in Swift.

Variables can change their values over time, while constants get an assigned value and keep that value throughout the execution of a program. Both variables and constants can store only one type of data.

Constants

Let's reproduce the Hello World example from Chapter 1 using a constant. Listing 2-1 shows how to store the string "`Hello World`" in a constant named `s1`.

Listing 2-1. Hello World Constant

```
let s1:String = "Hello World"
```

The first part of the constant declaration is the `let` keyword. The `let` keyword lets you know that you are working with a constant. The next part is the constant name `s1`. You will use the constant name `s1` to refer to this constant in your code from now on.

You also have the type declaration `:String`. The type declaration tells you what data type the constant stores. Since you used the type declaration `:String`, you know that you can store strings (a sequence of characters) in the constant `s1`.

The next part is the assignment operator `=`, which assigns a value to the constant `s1`. The value here is a string enclosed in quotes, `"Hello World"`.

If you use a playground to prototype the code here, you will see that it immediately reports the value of the `s1` constant on the right side of the screen.

You can reference a constant by using its name. To get the value of `s1`, you can just type in the constant name anywhere in your code. Try it right now by typing `s1` into your playground (see Listing 2-2).

Listing 2-2. Referencing Constants

```
s1
```

You will be using constants more as you learn about the capabilities of Swift.

Constants Are Immutable

Let's say you would rather have "Hello World" print as "Hello World!" (with an exclamation point). Since `s1` is a constant, you cannot simply change the value or this code would cause an error (see Listing 2-3).

Listing 2-3. Error Caused by Assigning a Value to a Constant

```
s1 = "Hello World!"
```

When you need to change a value when a program runs, you must use a variable.

Variables

Variables are mostly used like constants but with two key differences. The first is that variables use the `var` keyword instead of the `let` keyword when variables are being declared. The second difference is that variable values can change.

Listing 2-4 shows an example of a variable `s2` that can change value over time.

Listing 2-4. Variable Declaration

```
var s2:String = "Hello World"
```

As you can see in Listing 2-4, you use the `var` keyword to specify variables. Variables also don't require that you immediately assign a value to them, so you could have waited to assign the `"Hello World"` string to `s2`.

Variables Are Mutable

Since `s2` is a variable, you can change its value. So, if you wanted to say "Hello World" in Spanish instead, you could change the value of `s2` as shown in Listing 2-5.

Listing 2-5. Changing Variable Value

```
s2 = "Hola Mundo"
```

Now if you type `s2` into your playground, you will see the value "Hola Mundo" appear on the right.

Type Inference

In the previous examples, you clearly spelled out the data type for both the variables and the constants. However, Xcode can figure this out for you based on what value you assign to the variable or constant. This is called *type inference*. This means you could have omitted the `:String` from your declarations and instead use something like Listing 2-6.

Listing 2-6. Type Inference

```
var s3 = "Hallo Welt"
```

Data Types

Swift supports more than the `String` data type. You also work with numbers and booleans. Table 2-1 describes the common Swift data types.

Table 2-1. Swift Data Types

Data Type	Description
`String`	Sequence of characters
`Int`	Whole number
`Float`	Number with fractional component
`Double`	Bigger number with fractional component
`Bool`	True or false value

See Listing 2-7 for examples of how to use these data types.

Listing 2-7. Swift Data Types

```
let s:String = "Hey There"
let i:Int = -25
let f:Float = 3.14
let d:Double = 99.99
let b:Bool = true
```