Swift
im Detail

thomas **SILLMANN**

HANSER

Sillmann

Swift im Detail

Thomas Sillmann

Swift im Detail

HANSER

Der Autor:
Thomas Sillmann, Aschaffenburg
www.thomassillmann.de

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Lektorat: Sieglinde Schärl
Copy editing: Sandra Gottmann, Münster-Nienberge
Herstellung: Irene Weilhart
Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de
Umschlagrealisation: Stephan Rönigk
Gesamtherstellung: Kösel, Krugzell
Printed in Germany

Print-ISBN: 978-3-446-44294-8
E-Book-ISBN: 978-3-446-44423-2

Für Ela

– auf das, was war, und das, was sein wird.

Inhalt

1 Apples neue Programmiersprache: Swift

1.1 Willkommen bei Swift!

Das war schon eine ziemliche Überraschung im Juni 2014 zu Beginn von Apples alljährlicher Entwicklerkonferenz WWDC. Ich als Entwickler habe mit vielem gerechnet, allen voran mit der Vorstellung der neuen Betriebssystemversionen von OS X und iOS. Auch eine iWatch (inzwischen als Apple Watch bekannt) hätte ich mir damals bereits vorstellen können. Doch dass Apple dann nicht nur Unmengen neuer APIs und Frameworks für Entwickler aus dem Hut zaubert, sondern gleich noch eine komplett neue Programmiersprache vorstellt, hat wahrlich meine kühnsten Vorstellungen übertroffen und meine Kinnlade während der Präsentation weit herabsinken lassen. Immerhin bin ich damit nicht alleine, denn Swift war eine spannende und gänzlich unerwartete Vorstellung auf der WWDC 2014. Die Begeisterung und Faszination für diese Sprache ist seitdem ungebrochen und es gibt kaum einen iOS- und OS X-Entwickler, der sich nicht ausgiebig mit Apples neuer Programmiersprache auseinandersetzt. Nicht nur, dass Swift einfacher anzuwenden sein soll als das bisher von Apple präferierte Objective-C, nein, auch sollen zukünftig mit Swift entwickelte Apps bis zu 30 % schneller sein als ihre Objective-C-Pendants. Allein das ist Grund genug, dass sich nicht nur Einsteiger und Apple-Neulinge mit Swift intensiv auseinandersetzen.

1.2 Warum Swift?

Laut Apple begann die Entwicklung der Programmiersprache Swift im Jahr 2010. Hauptgrund für die Entwicklung von Swift dürfte unter anderem gewesen sein, dass Objective-C als bisherige Hauptsprache zur Programmierung für iOS und OS X den ein oder anderen Entwickler abschreckte. Nicht zuletzt aufgrund seiner Syntax war und ist Objective-C – gerade für Umsteiger von anderen Programmiersprachen – anfangs ein wenig befremdlich und schwer zu verstehen und zu erlernen. Eine Programmiersprache wie Java, die beispielsweise für die Entwicklung von Android-Apps genutzt wird und wesentlich weiter verbreitet ist, lockt da potenzielle App-Entwickler schon eher. Dieses Umstands war sich auch Apple bewusst, und so hat Swift auch einen klaren Grundsatz: Einfach, schnell, spaßig. Mit Swift

sollen Entwickler schnell und einfach eigene Anwendungen für iOS und OS X schreiben können und wahrlich Spaß an der Entwicklung haben.

Zu diesem Zweck hat Apple für Swift die Vorteile und Merkmale aus verschiedensten Programmiersprachen als Basis genommen und in Swift einfließen lassen. Neben Objective-C dienen so beispielsweise auch Ruby und Python als Vorlage für Swift.

Auch wenn selbstredend aktuell Objective-C bei den meisten App-Entwicklern weiter verbreitet ist, so ändert das aber nichts daran, dass Swift langfristig die bessere Alternative zu Objective-C sein dürfte. Allein die Tatsache, dass mit Swift entwickelte Apps bis zu 30 % schneller ausgeführt werden als in Objective-C geschriebene Anwendungen, ist eine deutliche Ansage. Auch macht die Einfachheit von Swift – verbunden mit allen Möglichkeiten und API-Zugriffen wie mit Objective-C auch – die Sprache ebenso für alteingesessene Entwickler attraktiv. Und ob Apple früher oder später nicht doch irgendwann den Objective-C-Hahn zudreht und nur noch Swift unterstützt, bleibt abzuwarten.

Sollten Sie jetzt ganz frisch mit der iOS- und/oder OS X-Entwicklung beginnen, kann ich Ihnen nur wärmstens empfehlen, sich voll und ganz auf Swift zu konzentrieren. Sie erlernen dann eine moderne Sprache mit moderner Architektur, die in den kommenden Jahren noch weitaus mehr Popularität und Anhänger gewinnen wird.

Und wenn Sie bereits mehrere Jahre mit Objective-C entwickeln (so wie ich auch), dann lassen Sie sich nichtsdestotrotz auf Swift ein. Starten Sie entweder ein neues Projekt mit Swift oder verwenden Sie Swift zur Erweiterung Ihrer bestehenden Anwendungen (denn Objective-C und Swift können problemlos gemischt innerhalb eines Projekts verwendet werden, aber dazu gleich mehr). Langfristig wird Swift *die* Sprache zur Entwicklung von iOS- und OS X-Apps sein, und umso früher Sie Ihre Expertise in diesem Bereich ausbauen und Erfahrungen sammeln, desto besser sind Sie auf alle Neuerungen und Erweiterungen der Sprache in Zukunft vorbereitet.

1.3 Swift und Objective-C

Wie bereits geschrieben, ist Swift zum aktuellen Zeitpunkt kein Ersatz, sondern eine neue Programmiersprache neben Objective-C zur Entwicklung von iOS- und OS X-Apps. Damit Sie sich aber nicht ausschließlich für eine der beiden entscheiden müssen, haben Sie die Möglichkeit, Swift- und Objective-C-Code parallel in einem Projekt zu verwenden. Zwar können Sie nicht innerhalb einer Quellcode-Datei Swift- und Objective-C-Code mischen, können aber in einem Projekt sowohl Swift- als auch Objective-C-Klassen verwenden, die dann jeweils Code der entsprechenden Programmiersprache enthalten (siehe Bild 1.1). Das erlaubt Ihnen, selbst bereits bestehende und in Objective-C geschriebene Anwendungen um neue Swift-Klassen und -Funktionen zu erweitern.

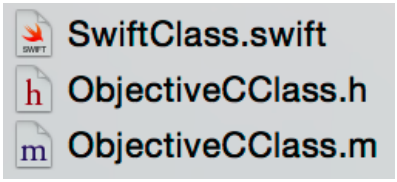

Bild 1.1
Ja, sie verstehen sich; Objective-C- und Swift-Klassen können problemlos parallel in einem Projekt verwendet werden.

Mehr zur parallelen Verwendung von Swift und Objective-C erfahren Sie im Kapitel 5 „Swift, Cocoa und Objective-C“.

1.4 Voraussetzungen für die Swift-Entwicklung

1.4.1 Xcode

Sie können Swift momentan ausschließlich für die Entwicklung von Apps für iOS und OS X verwenden. Grundlage für die Entwicklung solcher Apps ist Xcode – Apples hauseigene Entwicklungsumgebung (siehe Bild 1.2). Xcode ist ausschließlich auf dem Mac verfügbar und läuft in der aktuellen Version 6 ab OS X 10.9.4. Neben dem Editor und weiteren Tools für die Software-Entwicklung enthält Xcode auch alle benötigten SDKs sowie den Compiler, um Anwendungen in Swift für iOS und OS X zu schreiben und auszuführen.

Bild 1.2
Apples Entwicklungsumgebung Xcode enthält alles, was Sie für die Entwicklung von Anwendungen mit Swift benötigen.

Die Installation von Xcode gestaltet sich sehr simpel: Sie finden die Anwendung über den Mac App Store, der inzwischen fester Bestandteil des Betriebssystems OS X des Mac ist. Darüber können Sie Xcode einfach herunterladen und installieren (siehe Bild 1.3). Aufgrund seiner Größe von etwas mehr als zwei Gigabyte kann es aber sein, dass der Download – je nach vorhandener Internetverbindung – eine gewisse Zeit in Anspruch nimmt.

Bild 1.3 Xcode können Sie direkt aus dem App Store heraus auf Ihrem Mac installieren.

Neben dem Editor und den SDKs bringt Xcode unter anderem auch iOS-Simulatoren, Apples Entwickler-Dokumentation, eine integrierte Versionsverwaltung und weitere hilfreiche Tools mit; eine ausführliche Vorstellung der IDE würde den Rahmen dieses Buches bei Weitem sprengen. Alle notwendigen Grundlagen, um die in diesem Buch beschriebenen Beispiele nachzuvollziehen und mit der Entwicklung von Apps mit Swift zu beginnen, werde ich aber selbstverständlich im Detail vorstellen und erläutern. Dazu finden Sie in Kapitel 6, „Swift und Xcode“, weitere Informationen zur Installation und Arbeit mit Xcode im Zusammenspiel mit Apples neuer Programmiersprache Swift.

1.4.2 Mac

Wie Sie sehen, ist auch ein Mac Voraussetzung, um mit Swift entwickeln zu können, denn Apple stellt seine SDKs ausschließlich für die eigene Plattform bereit. Welchen Mac Sie letztlich für die Arbeit mit Xcode verwenden, bleibt gut und gerne Ihren persönlichen Vorlieben überlassen; leistungsstark genug sind alle verfügbaren Modelle (siehe Bild 1.4). Lediglich beim Arbeitsspeicher sollten es nach Möglichkeit mindestens 8 GB sein; das erleichtert das Ausführen Ihrer Apps ungemein und macht die Arbeit mit Xcode komfortabler.

Bild 1.4 Die Wahl des passenden Mac zur Entwicklung mit Swift bleibt komplett Ihren persönlichen Vorlieben überlassen.

1.5 Swift-Ressourcen

Ich darf Ihnen an dieser Stelle gratulieren: Eine der wichtigsten Ressourcen für Ihre tägliche Arbeit mit Swift halten Sie bereits in Händen. ☺ Nein im Ernst, natürlich werden Sie mithilfe dieses Buches die Entwicklung von Apps mittels Swift erlernen und in die Lage versetzt, eigene Anwendungen mit Swift zu schreiben und Programmierprobleme selbstständig zu lösen. Aber es ist eine Tatsache, dass dieses Buch nicht jeden Teilbereich vollumfänglich abdecken kann und sich darüber hinaus Swift im Laufe der Jahre auch noch weiterentwickeln und möglicherweise verändern wird. Aus diesem Grund ist es wichtig zu wissen, welche weiteren Ressourcen Ihnen zur Verfügung stehen und welche Sie für Ihre tägliche Arbeit nutzen können.

1.5.1 Apples Entwickler-Dokumentation

Am wichtigsten hierbei ist Apple mit seiner ausführlichen und umfangreichen Entwickler-Dokumentation. Diese Dokumentation ist sowohl online für registrierte Apple-Entwickler als auch über Apples Entwicklungsumgebung Xcode verfügbar. Sie enthält nicht nur die Beschreibung aller verfügbaren APIs für die iOS- und OS X-Entwicklung, nein, sie enthält darüber hinaus auch Code-Beispiele, Schritt-für-Schritt-Anleitungen und stets die topaktuellsten Informationen zu Swift.

Als registrierter Apple-Entwickler können Sie die Online-Dokumentation auf Apples Entwicklerplattform unter *https://developer.apple.com* abrufen (siehe Bild 1.5). Dort erhalten Sie auch Zugang zu Vorabversionen von Xcode, iOS und OS X, müssen dazu aber einem der kostenpflichtigen Apple Developer Programs beitreten.

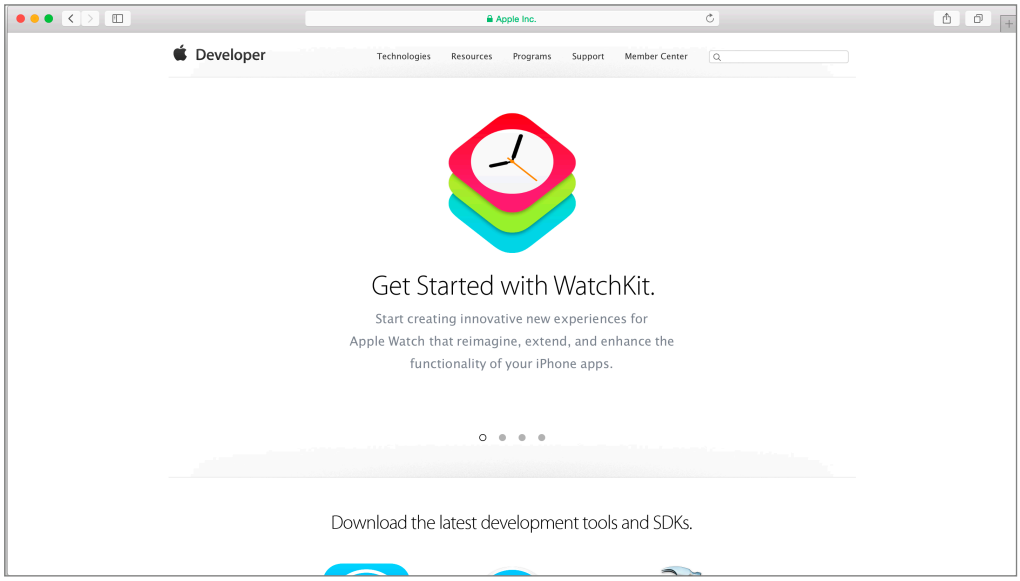

Bild 1.5 Der Ausgangspunkt für jeden Apple-Entwickler: Apples Developer-Portal.

Entwickler-Dokumentation herunterladen

Wie beschrieben ist die Entwickler-Dokumentation von Apple nicht nur online, sondern auch offline über die IDE Xcode verfügbar. Dazu müssen Sie diese aber erst einmal über Xcode herunterladen, denn standardmäßig greift auch Xcode auf die Online-Version der Dokumentation zu.

Gehen Sie zum Download der Entwickler-Dokumentation wie folgt vor:

1. Starten Sie Xcode
2. Wechseln Sie in die Einstellungen über das Menü XCODE → PREFERENCES... oder verwenden Sie das Tastaturkürzel CMD + ,
3. Wechseln Sie in den Reiter DOWNLOADS

Dort werden Ihnen im Abschnitt „Documentation“ alle verfügbaren Dokumentationen angezeigt (siehe Bild 1.6). Rechts daneben befindet sich ein Download-Button, durch den Sie den Download einer Dokumentation anstoßen können.

Damit Xcode automatisch immer den aktuellsten Stand der Entwickler-Dokumentation herunterlädt und somit offline vorhält, können Sie den Haken bei *Check for and install updates automatically* setzen. Damit prüft Xcode bei jedem Start, ob eine aktualisierte Version der Dokumentation vorhanden ist, und lädt diese bei Bedarf automatisch im Hintergrund herunter.

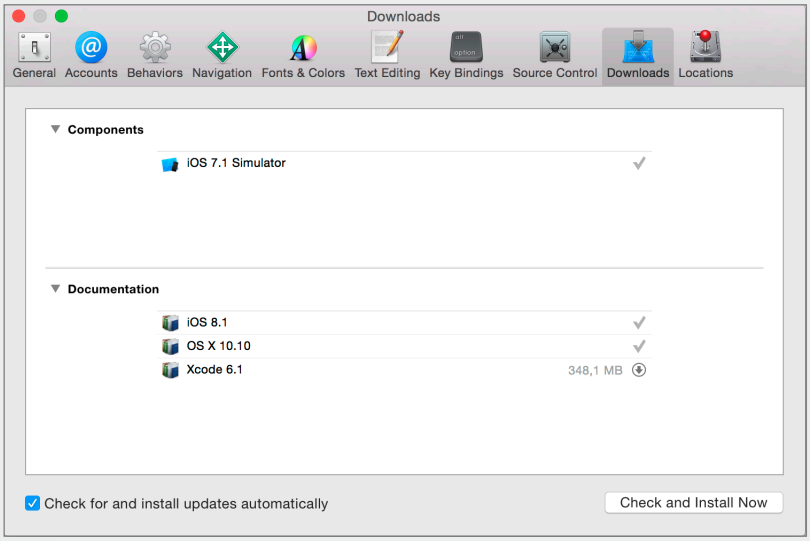

Bild 1.6 Xcode erlaubt den kompletten Download von Apples Entwickler-Dokumentation.

1.5.2 Swift-Blog

Mit der Vorstellung von Swift startete Apple auf seiner Entwickler-Website einen eigenen Swift-Blog. Dieser informiert einerseits über die Fortschritte und die Entwicklung der Sprache, zeigt aber auch typische Design Patterns und Best Practices auf. Der Blog ist also immer eine großartige Informationsquelle, um Neuigkeiten über Swift zu erfahren. Darüber hinaus erlaubt der Blog den direkten Zugriff auf wichtige Entwickler-Ressourcen wie passende WWDC-Sessions zu Swift, die Entwicklungsumgebung Xcode sowie Code-Beispiele von Apple zu Swift.

Der Swift-Blog ist über den URL *https://developer.apple.com/swift/* erreichbar (siehe Bild 1.7).

Bild 1.7 Der Swift-Blog von Apple ist immer eine lohnenswerte Anlaufstelle für Entwickler.

1.5.3 Code-Beispiele des Autors

Im Zuge meines vorangegangenen Buchprojekts „Apps für iOS 8 professionell entwickeln“ habe ich auf meiner Website einen eigenen Bereich für diverse vorgefertigte Klassen eingefügt. Dabei handelt es sich um generische Klassen, die Sie so problemlos und in der Regel ohne Anpassungen am Code in Ihren eigenen Projekten verwenden können. All diese Klassen habe ich sowohl in Objective-C als auch in Swift geschrieben und sie stehen auf der Website zum Buch zum Download bereit. Sie können diese Klassen einerseits dazu nutzen, um einmal einen Blick auf praktisch umgesetzten Swift-Code zu werfen sowie die ein oder andere dieser Klassen möglicherweise auch in eigenen Projekten einzusetzen. Darüber hinaus können Sie damit einen direkten Vergleich zwischen Objective-C- und Swift-Code anstellen, indem Sie sich beide Sprachversionen zu einer Klasse herunterladen und anschließend vergleichen.

thomassillmann.de/swift-buch

Schauen Sie doch einfach mal auf meiner Website zum Buch vorbei, hier finden Sie die Code-Beispiele sowie zusätzliche Informationen zum Thema.

1.5.4 Das Internet

Wenn Sie kein Neuling in der App-Entwicklung (oder auch allgemein in der Software-Entwicklung) sind, brauche ich Ihnen das sicherlich gar nicht zu erzählen, doch für alle Neulinge tue ich es trotzdem: Sind alle bereits vorgestellten Ressourcen erschöpft und Sie kommen partout bei einem Problem nicht weiter, hilft Ihnen womöglich das Wissen des World Wide Web weiter. Eine passende Anfrage bei der Suchmaschine Ihrer Wahl liefert Ihnen meist eine gute Auswahl passender Tutorials und Hilfestellungen.

Häufig werden Sie dabei auch sicherlich auf das Portal Stack Overflow (*stackoverflow.com*) stoßen. Dabei handelt es sich um eine Plattform für Software-Entwickler aus allen Bereichen, die dort Fragen einstellen können, die sodann andere Mitglieder aus der Stack Overflow-Community beantworten können.

Auch darüber hinaus werden Sie in den Weiten des Internets viele verschiedene Informationen finden. Gerne können Sie sich auch von meinen Blog-Beiträgen zum Thema App-Entwicklung inspirieren lassen, schauen Sie dazu einfach auf *thomassillmann.de* vorbei.

2 Grundlagen der Programmierung

Hier wären wir also, in der spannenden Welt der Programmierung mit Swift! Dieses Kapitel befasst sich mit den essenziellen Grundlagen der Programmiersprache. Sie lernen Variablen und Konstanten, Abfragen und Schleifen, Kommentare sowie die sogenannten Fundamental Types wie String und Array kennen. Dieses Wissen benötigen Sie, um darauf aufbauend im nächsten Schritt eigene Klassen und Methoden zu erstellen und damit effektiv mit Swift zu programmieren. Kapitel 3, „Objektorientierte Programmierung mit Swift", setzt sich dann intensiv mit eben diesen Themen wie Klassen, Funktionen, Initialisierung und Speicherverwaltung auseinander.

Bevor wir uns mit den Details der Programmiersprache Swift auseinandersetzen, gibt es ein paar Punkte, die ich vorab besprechen und vorstellen möchte und die sich durch alle weiteren Aspekte und Funktionen von Swift ziehen.

Befehlsabschluss mit oder ohne Semikolon

Eine Code-Zeile in Swift müssen Sie nicht zwingend – wie beispielsweise in Objective-C – mit einem Semikolon **;** abschließen, können es aber tun. Generell verzichtet Apple in all seinen bisherigen Beispielen und Ausführungen zur Sprache in Gänze auf das abschließende Semikolon am Ende eines Befehls und genau so werde ich es auch bei allen Beispielen und Erläuterungen in diesem Buch handhaben. Das Semikolon ist lediglich dann Voraussetzung, wenn Sie mehrere Befehle hintereinander in eine Zeile schreiben; dann müssen Sie das Semikolon nach Abschluss jedes Befehls setzen.

println()

In vielen Beispielen in diesem Buch wird Ihnen der Befehl `println()` begegnen. Dieser sorgt dafür, dass ein Wert oder eine Variable, die zwischen die beiden runden Klammern gesetzt wird, auf der Konsole der Entwicklungsumgebung ausgegeben wird. Damit können Sie schnell und einfach Informationen auslesen oder die Werte von Variablen abfragen. In Beispielen dient dieser Befehl dazu, die generierte Ausgabe des Codes durch Swift zu verdeutlichen und zu veranschaulichen.

Fundamental Types

Swift bringt von Haus aus eine Handvoll sogenannter Fundamental Types mit, die Sie für die Entwicklung Ihrer Anwendung in Swift verwenden können. Wenn Sie nicht gerade mit Klassen und Objekten arbeiten (dazu später im Kapitel 3 „Objektorientierte Programmierung mit Swift“ mehr), werden Sie es immer mit den existierenden Fundamental Types zu tun haben, wenn Sie mit Variablen und Konstanten arbeiten. All diese Fundamental Types werde ich an entsprechender Stelle im Abschnitt 2.4 „Fundamental Types“ im Detail vorstellen, nichtsdestoweniger gebe ich Ihnen hier bereits einmal eine Übersicht aller verfügbaren und existierenden Fundamental Types in Swift (siehe Tabelle 2.1):

Tabelle 2.1 Fundamental Types in Swift

Fundamental Type	Beschreibung	Beispiele
Int	Ein Integer (Int) stellt eine Ganzzahl dar.	19 99
Float	Bei Float handelt es sich um eine Fließkommazahl	19.99 49.94
Double	Auch bei Double handelt es sich um eine Fließkommazahl, allerdings ist der Wertebereich von Double deutlich größer als der von Float; entsprechend belegt ein Double auch mehr Speicherplatz im System als ein Float.	99.19 94.49
Bool	Bei Bool handelt es sich um einen Wahrheitswert, ein Bool ist entweder wahr oder falsch (true/false).	true false
String	Ein String repräsentiert eine Zeichenkette.	„Mein Name ist Thomas Sillmann.“
Array	In einem Array können mehrere Werte und Objekte abgelegt werden. Das Array erlaubt dann den Zugriff auf die Werte und Objekte, die es hält. Ein Array kann dabei beliebige Typen von Werten und Objekten beinhalten.	[„Erster Wert des Arrays“, „Zweiter Wert des Arrays“]
Dictionary	Ein Dictionary hält mehrere Werte und Objekte ähnlich wie ein Array, allerdings ist jeder Wert und jedes Objekt einem einzigartigen Schlüssel innerhalb des Dictionaries zugeordnet. Anhand dieses Schlüssels können dann gezielt Werte ausgelesen, abgefragt und verändert werden.	[„Schlüssel 1“: „Wert für Schlüssel 1“, „Schlüssel 2“: „Wert für Schlüssel 2“]

Playgrounds

Mit der Einführung von Swift und dazu der neuen Version 6 der Entwicklungsumgebung Xcode hat Apple die sogenannten Playgrounds eingeführt. Dabei handelt es sich um Dateien, in die Sie ganz normal und wie gewohnt Ihren Code schreiben können, sie sind aber – wie der Name bereits vermuten lässt – zum Experimentieren und Ausprobieren gedacht. Mit Playgrounds können Sie auf die Schnelle einmal eine neue Funktion testen oder neue Dinge durch Ausprobieren erlernen. Gerade deshalb sind Playgrounds auch für Swift-Neulinge eine großartige Sache. So haben Sie die Möglichkeit, alle Beispiele aus diesem Kapitel einfach nachzuvollziehen, indem Sie sie in einem Playground abtippen. Xcode zeigt Ihnen die Ergebnisse Ihrer Code-Befehle immer rechts in einem separaten Bereich der entsprechenden Zeile an (siehe Bild 2.1).

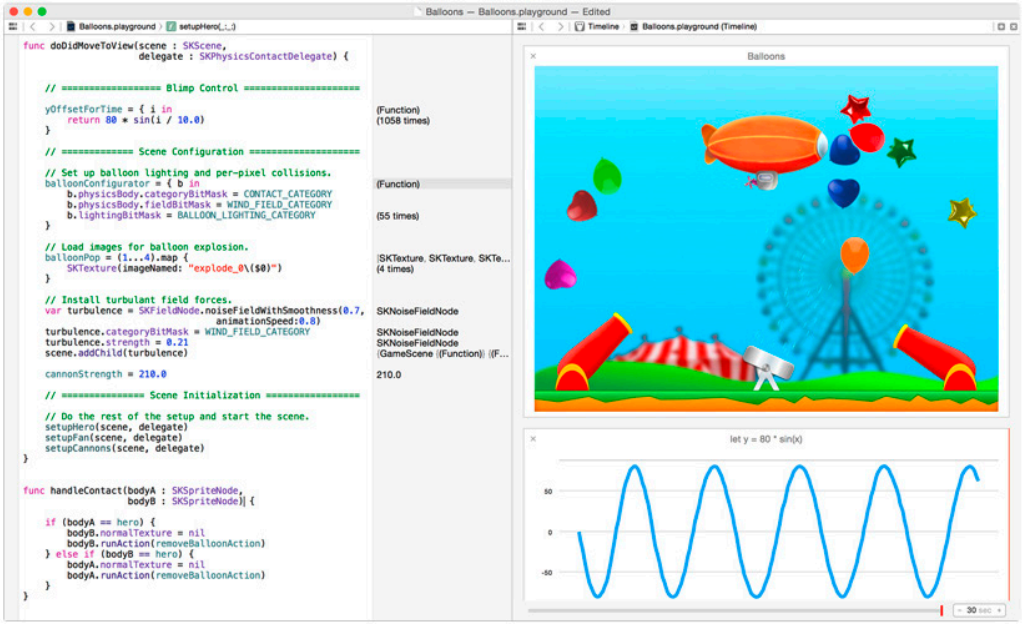

Bild 2.1 Playgrounds erlauben das Experimentieren mit Swift und das Ausprobieren verschiedener Funktionen. (Quelle: *https://developer.apple.com/swift/*)

Einen neuen Playground erstellen Sie über Xcode wie folgt: Starten Sie Xcode und klicken Sie auf die Schaltfläche **Get started with a playground** von Xcodes Startfenster aus (siehe Bild 2.2).

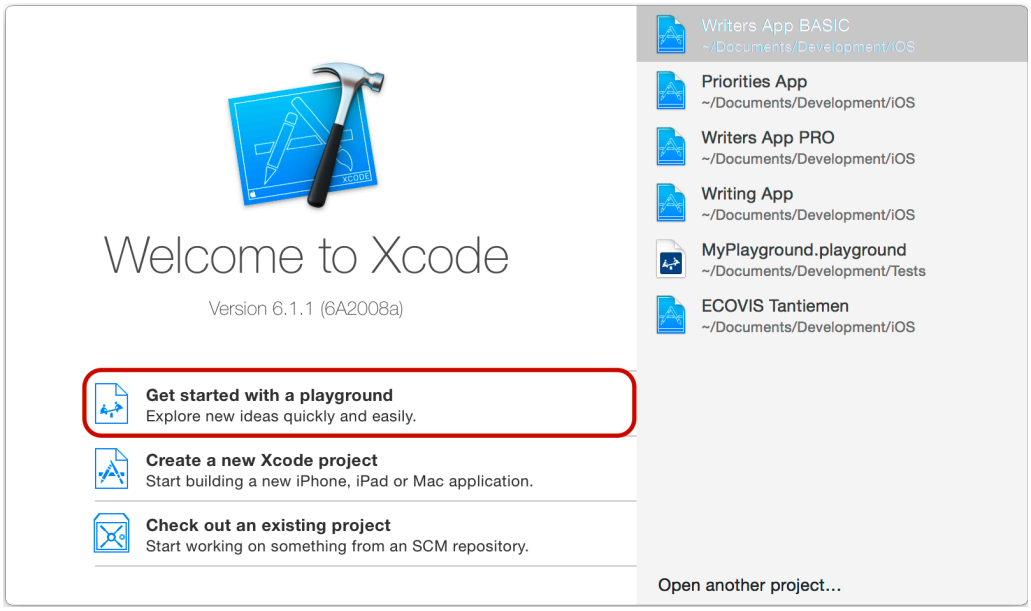

Bild 2.2 Ein neuer Playground kann direkt aus Xcodes Startfenster heraus erstellt werden.

In einem zweiten Schritt müssen Sie nur noch einen beliebigen Namen für die Playground-Datei vergeben sowie Ihre gewünschte Ziel-Plattform für Ihre Experimente im Playground angeben (siehe Bild 2.3). Das ist wichtig, weil iOS und OS X je unterschiedliche Frameworks und APIs besitzen und der Playground wissen muss, welche Funktionen er implementiert und zugänglich macht.

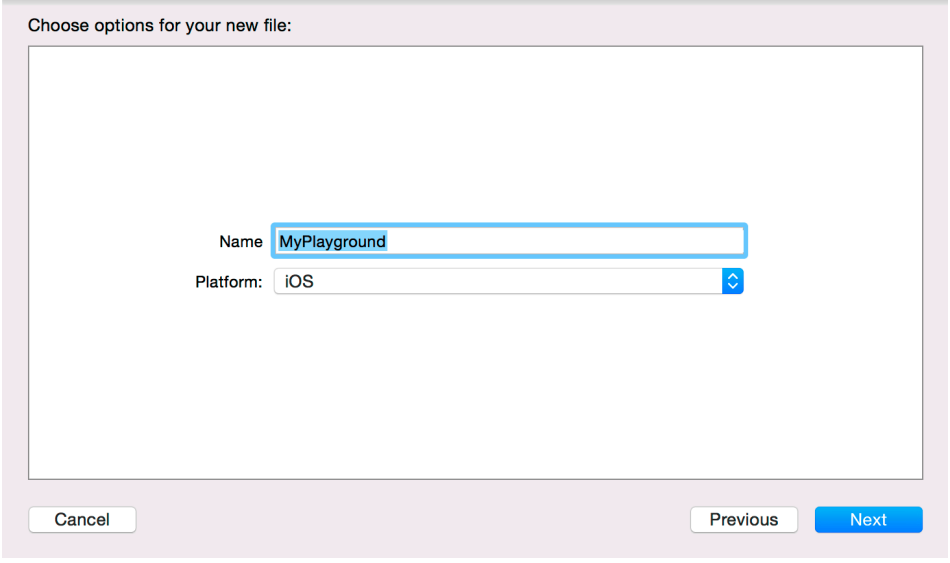

Bild 2.3 Sie können einen beliebigen Namen für den Playground vergeben.

Nach einem abschließenden Klick auf NEXT müssen Sie noch einen beliebigen Speicherort für die Playground-Datei auswählen und haben es geschafft. In der so erzeugten Playground-Datei können Sie nun nach Belieben mit Swift experimentieren, erste Erfahrungen sammeln oder die Lösung eines spezifischen Programmierproblems in einem Playground Schritt für Schritt angehen.

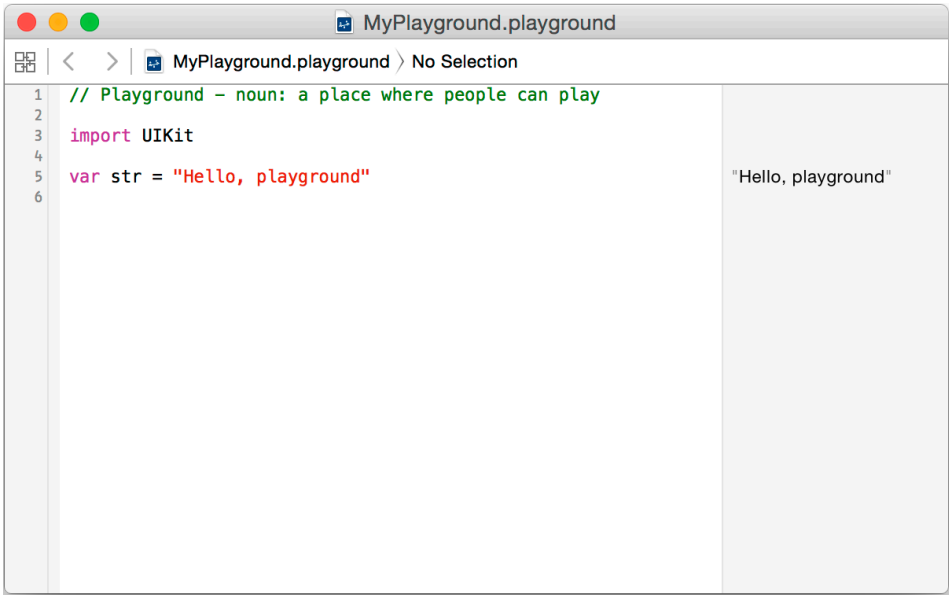

Bild 2.4 Der neu erstellte Playground enthält gleich einen ersten kleinen Code-Schnipsel mit Swift.

Playgrounds unterstützen ausschließlich Swift

Apple hat die Playgrounds ausschließlich für Swift konzipiert, eine Verwendung mit anderen Programmiersprachen – auch Objective-C – ist nicht möglich.

2.1 Variablen und Konstanten

Variablen dienen Ihnen in der Programmierung dazu, Werte und Objekte (zwischen) zu speichern, auf sie zuzugreifen und sie – im Falle von Variablen – zu verändern.

Eine einfache Variable erstellen Sie in Swift mithilfe des Schlüsselworts `var`, gefolgt von einem von Ihnen beliebig festlegbaren Variablennamen und der Zuweisung eines Werts für diese Variable mithilfe des Gleichheitszeichens =:

```
var myVariable = 19
```

Damit ist die Variable erstellt und deklariert und kann nun über den von uns festgelegten Variablennamen `myVariable` angesprochen und genutzt werden. Darüber hinaus haben wir ihr die Ganzzahl `19` als Wert zugewiesen. Den zugewiesenen Wert können wir aber jederzeit ändern, indem wir über den festgelegten Variablennamen erneut auf die Variable zugreifen und einen neuen Wert setzen:

```
var myVariable = 19
myVariable = 99
```

In diesem Fall wurde in der ersten Zeile die Variable mit dem Namen `myVariable` erstellt und ihr der Zahlenwert `19` zugewiesen. Anschließend wurde die Variable mithilfe ihres Variablennamens erneut aufgerufen und ihr stattdessen der Zahlenwert `99` zugewiesen.

Um eine Konstante zu erstellen, gehen Sie genauso vor wie bei der Erstellung einer Variablen, verwenden aber statt des Schlüsselworts `var` das Schlüsselwort `let`:

```
let myConstant = 19
```

Wichtig bei Konstanten ist, dass Sie diesen direkt beim Erstellen einen passenden Wert zuweisen müssen. Denn während Variablen jederzeit geändert werden können (wie wir an unserem vorherigen Beispiel gesehen haben), sind Konstanten unveränderlich; ihr einmalig zugewiesener Wert bei der Deklaration ändert sich während der gesamten Laufzeit nicht mehr. Folgender Code würde also einen Compiler-Fehler verursachen:

```
let myConstant = 19
myConstant = 99 // Error: Cannot assign to ‚let' value ‚myConstant'
```

Warum nicht einfach immer Variablen verwenden?

Möglicherweise fragen Sie sich an dieser Stelle, warum Sie nicht einfach immer Variablen verwenden; dann kann das im letzten Beispiel aufgeführte Problem ja überhaupt nicht erst auftreten? Tja, so einfach ist es leider nicht. Theoretisch wäre ein solches Vorgehen denkbar, aber das hat dann nichts mit gutem und sauberem Code zu tun.

In der Programmierung werden Sie des Öfteren Werte und Objekte benötigen, die Sie einmal erzeugen und anschließend – ohne sie jemals zu verändern – an mehreren Stellen wiederverwenden. Haben Sie einen solchen Wert beziehungsweise ein solches Objekt mittels `let` als Konstante deklariert, haben Sie im Code auch überhaupt keine Chance, jemals eine Änderung für diesen Wert beziehungsweise dieses Objekt vorzunehmen. Sie stellen also somit sicher, dass Ihr Code das tut, was er soll, und diese Konstante nur verwendet, nicht aber verändert.

Denn generell gilt in Swift: Wenn Sie einen Wert oder ein Objekt speichern, das nicht mehr verändert werden soll, dann deklarieren Sie es als Konstante mittels `let`. Nicht nur kann dann der Compiler prüfen, dass dieser Wert beziehungsweise dieses Objekt auch tatsächlich nicht verändert wird, nein, Sie machen es auch dem System leichter, da es weiß, dass sich dieser Wert beziehungsweise dieses Objekt an keiner Stelle mehr verändern wird.

Und glauben Sie mir: Gerade in Swift werden Sie sehr oft mit Konstanten arbeiten!

2.1.1 Type Inference und Type Annotation

Bei der eben gezeigten Variante zum Erstellen von Variablen und Konstanten hat bisher aber immer eine Information gefehlt: Um welchen Typ handelt es sich denn überhaupt bei den erstellten Variablen und Konstanten? Wir haben bereits zuvor im Abschnitt 2.4 „Fundamental Types“ die Typen kennengelernt, mit denen Swift von Haus aus arbeiten kann und die wir für all unsere Variablen und Konstanten problemlos verwenden können.

Nehmen wir noch einmal eines der Code-Beispiel von eben:

```
var myVariable = 19
```

Wir erstellen hier also eine Variable mit Namen `myVariable` und weisen ihr den Wert `19` zu. Laut unserer Übersicht der Fundamental Types in Swift handelt es sich dabei um einen Int, eine Ganzzahl. Zwar haben wir diese Information nicht explizit angegeben, doch ist aufgrund des Werts, den wir dieser Variablen zuweisen, davon auszugehen.

Und genau das tut auch Swift. Wird kein expliziter Typ angegeben, setzt Swift selbst den Typ, den es für diesen Wert als passend erachtet; dies wird als Type Inference bezeichnet. Unsere Variable `myVariable` ist nach dieser Zuweisung also ein Int. Würden wir anschließend versuchen, dieser Variablen einen Wert eines anderen Typs – zum Beispiel einen Wert vom Typ String – zuzuweisen, würden wir einen Compiler-Fehler verursachen:

```
var myVariable = 19
myVariable = "Das ist ein String" // Error: Type Ìnt` does not conform
to protocol StringLiteralConvertible`
```

Die vom Compiler generierte Fehlermeldung zeigt uns, dass wir `myVariable` keinen String zuweisen können; schließlich handelt es sich bei der Variablen ja um einen Int. Auch wenn (oder besser gesagt gerade weil) wir das nie explizit angegeben haben, geht Swift aufgrund des Werts, den wir der Variablen als Erstes zugewiesen haben, schlicht und einfach davon aus.

Wir können aber auch selbst den Typ einer Variablen oder Konstanten festlegen; dabei handelt es sich dann um die sogenannte Type Annotation. Dazu muss der Typ bei der Erstellung unserer Variablen oder Konstanten mit angegeben werden:

```
var myIntegerVariable: Int = 19
let myStringConstant: String = "Das ist ein String"
```

Dazu wird der Typ nach dem Variablennamen getrennt durch einen Doppelpunkt angegeben. Dadurch legen *wir* fest, welchen Typen diese Variable besitzt und welche Zuweisungen erlaubt sind. Das macht vor allen Dingen dann Sinn, wenn bei der Erstellung einer Variablen nicht sofort ein Wert zugewiesen werden soll, sondern erst später im Programmverlauf. Dann bleibt uns gar keine andere Wahl, als einen genauen Typ für diese Variable festzulegen: