

Walter Saumweber

SCHNELLEINSTIEG SWIFT 2



160 SEITEN

IN NEUN PRAXISBEZOGENEN
SCHRITTEN DIE PROGRAMMIER-
SPRACHE FÜR IOS-APPS ERLERNEN.

FRANZIS

Walter Saumweber hat langjährige Erfahrung als Entwickler, Berater und Dozent. Er veröffentlicht regelmäßig Artikel in IT-Fachzeitschriften und ist Autor von zahlreichen Computerfachbüchern. Seine Tätigkeitsschwerpunkte sind die Realisierung von Unternehmenslösungen in Client/Server-Umgebungen und Workflow-Anwendungen.

Walter Saumweber

SCHNELLEINSTIEG SWIFT 2

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Alle Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, dass sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung etwaiger Fehler sind Verlag und Autor jederzeit dankbar. Internetadressen oder Versionsnummern stellen den bei Redaktionsschluss verfügbaren Informationsstand dar. Verlag und Autor übernehmen keinerlei Verantwortung oder Haftung für Veränderungen, die sich aus nicht von ihnen zu vertretenden Umständen ergeben. Evtl. beigefügte oder zum Download angebotene Dateien und Informationen dienen ausschließlich der nicht gewerblichen Nutzung. Eine gewerbliche Nutzung ist nur mit Zustimmung des Lizenzinhabers möglich.

© 2016 Franzis Verlag GmbH, 85540 Haar bei München

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Das Erstellen und Verbreiten von Kopien auf Papier, auf Datenträgern oder im Internet, insbesondere als PDF, ist nur mit ausdrücklicher Genehmigung des Verlags gestattet und wird widrigenfalls strafrechtlich verfolgt.

Die meisten Produktbezeichnungen von Hard- und Software sowie Firmennamen und Firmenlogos, die in diesem Werk genannt werden, sind in der Regel gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden. Der Verlag folgt bei den Produktbezeichnungen im Wesentlichen den Schreibweisen der Hersteller.

Autor: Walter Saumweber

Programmleitung: Dr. Markus Stäuble

Satz: DTP-Satz A. Kugge, München

art & design: www.ideehoch2.de

Druck: CPI-Books

Printed in Germany

ISBN 978-3-645-60256-3

INHALTSVERZEICHNIS

1	XCODE EINRICHTEN UND VERWENDEN	8
	1.1 Entwickler-Tools installieren	8
	1.2 Ein Projekt anlegen	9
	1.3 Dem Projekt Steuerelemente hinzufügen	13
	1.4 IBOutlet und IBAction	14
	1.5 Playgrounds verwenden	18
2	SWIFT-BASICS	20
	2.1 Einzeilige und mehrzeilige Kommentare	20
	2.2 Literale, Rechenoperatoren, Anweisungen	21
	2.3 let und var	23
	2.4 Bezeichnerwahl.	29
	2.5 Elementare Datentypen.	32
	2.5.1 Integer-Datentypen.	32
	2.5.2 Float und Double	35
	2.5.3 Der boolesche Datentyp	37
	2.5.4 Wissenswertes über Strings	38
	2.5.5 Der Datentyp »Character«	41
	2.6 Implizite und explizite Typkonvertierungen	44
3	WEITERE OPERATOREN, INITIALIZER, OPTIONALS	48
	3.1 Der Modulo-Operator	48
	3.2 Inkrement- und Dekrementoperator	49
	3.3 Zusammengesetzte Zuweisungsoperatoren	51
	3.4 Bereichsoperatoren.	52
	3.5 Vergleichsoperatoren	53
	3.6 Logische Operatoren.	54
	3.7 Initializer.	58

3.8	Optionals	59
3.9	Der Operator »??«	64
4	KONTROLLSTRUKTUREN	66
4.1	if, if-else, else-if	66
4.2	switch-Anweisung	69
4.3	while-Schleife	73
4.4	repeat-while-Schleife	75
4.5	for-Schleife	76
4.6	break und continue	79
4.7	Mit Sprungmarken versehene Schleifen	80
5	ARRAYS, DICTIONARIES UND TUPEL	82
5.1	Arrays definieren und verwenden	82
5.2	for-in-Schleife	85
5.3	Operationen mit Arrays durchführen	87
5.4	Dictionaries	91
5.5	Daten in einem Tupel zusammenfassen	99
6	FUNKTIONEN IN SWIFT	102
6.1	Funktionen definieren und aufrufen	102
6.2	Funktionen mit mehreren Rückgabewerten definieren	108
6.3	Benannte und unbenannte Parameter	110
6.4	Standardwerte für Parameter festlegen	112
6.5	Beliebig viele Werte an eine Funktion übergeben	114
6.6	Mit »var« und »inout« deklarierte Parameter	115

7	FUNKTIONSTYPEN UND CLOSURE-AUSDRÜCKE	120
	7.1 Funktionstypen	120
	7.2 Funktionen als Parameter von Funktionen.	122
	7.3 Funktionen als Rückgabewerte.	124
	7.4 Funktionen verschachteln	124
	7.5 Closures definieren und verwenden	127
	7.6 Trailing Closures und andere Kurzschreibweisen	131
8	MIT SWIFT OBJEKTORIENTIERT PROGRAMMIEREN	134
	8.1 Strukturen definieren	134
	8.2 Objekte erstellen.	135
	8.3 Mutating-Methoden.	137
	8.4 Eigenschaften initialisieren	137
	8.5 Klassen sind Referenztypen.	141
	8.6 Enumerationen	142
9	SPEZIELLE OOP-KONZEPTE	146
	9.1 Lazy Properties	146
	9.2 Computed Properties	147
	9.3 Statische Eigenschaften und Methoden.	150
	9.4 Zugriffsmodifizierer	152
	9.5 Vererbung.	153
	STICHWORTVERZEICHNIS	156

In diesem Kapitel geht es um die Grundlagen der Programmiersprache Swift. Sie lernen die wichtigsten Syntaxregeln sowie die Verwendung von Konstanten, Variablen und Datentypen kennen.

2.1 Einzeilige und mehrzeilige Kommentare

Lassen Sie uns gleich an den im letzten Kapitel vorgestellten Playground anknüpfen. Die erste Zeile des von Xcode eingefügten Codes lautet dort:

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

Dass dies keine Anweisung darstellt und somit nichts mit dem eigentlichen Swift-Programmcode zu tun hat, liegt auf der Hand. Vielmehr handelt es sich um einen Kommentar – hier mit dem Hinweis, dass es sich bei einem Playground um einen »Spielplatz« handelt. Kommentare werden bei der Übersetzung des Quellcodes ignoriert, sie haben daher keinerlei Einfluss auf das Laufzeitverhalten der fertigen App und sie bringen auch sonst keine Nachteile mit sich. Es handelt sich lediglich um Hinweise, die der App-Entwickler in den Quelltext einfügen kann. Dementsprechend sind die Adressaten von Kommentaren nicht etwa die späteren Benutzer der fertigen App, sondern die Leser des Quellcodes – also der Programmautor selbst oder andere Entwickler, die den Code eventuell zu einem späteren Zeitpunkt überarbeiten. Der gezielte Einsatz von Kommentaren kann ihnen dabei behilflich sein, sich im Code – den sie ja eventuell nicht selbst geschrieben haben – schneller zurechtzufinden. Sie können Kommentare aber z. B. auch als Gedankenstütze einsetzen, etwa wenn Sie beabsichtigen, an einer bestimmten Codestelle zu einem späteren Zeitpunkt noch Verbesserungen durchzuführen.

Swift unterstützt einzeilige und mehrzeilige Kommentare. Einzeilige Kommentare werden mit einem doppelten Slash (//) eingeleitet. Der Text, der dem // nachfolgt, wird bis zum Zeilenende als Kommentar angesehen. In der oben genannten Kommentarzeile gehört der Doppelpunkt nach den beiden Schrägstrichen folglich nicht zum Kommentarzeichen, sondern zum nachfolgenden Kommentar. Genauso gut hätte man daher auch

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

notieren können, und sogar das Leerzeichen nach den beiden Schrägstrichen ist nicht obligatorisch.

Möchten Sie einen ganzen Absatz als Kommentar auszeichnen, brauchen Sie nicht jede Zeile mit einem doppelten Schrägstrich zu beginnen, sondern Sie können die Zeichen `/*` und `*/` verwenden. Das Zeichen `/*` steht für den Anfang, das Zeichen `*/` für das Ende eines Kommentars. Der gesamte Text, der sich zwischen diesen beiden Zeichen befindet, wird als Kommentar angesehen.

```
001 /* Dies ist ein  
002 Kommentar, der sich  
003 über mehrere  
004 Zeilen erstreckt */
```

TIPP:

Apropos »in Kommentar setzen«: Dies ist ein probates Mittel, wenn Sie für einen Testlauf bestimmte Anweisungen beim Kompilieren nicht berücksichtigen wollen. Setzen Sie einfach das Kommentarzeichen `/*` an den Anfang und `*/` an das Ende des auszukomentierenden Codestücks. Der Compiler – das ist die integrierte Software, die den Swift-Quellcode in ausführbaren Code übersetzt – wird die enthaltenen Anweisungen daraufhin ignorieren. Möchten Sie später, dass diese Anweisungen beim Programmlauf wieder ausgeführt werden, entfernen Sie die beiden Kommentarzeichen einfach.



Swift erlaubt es auch, Kommentare zu verschachteln (`/* ... /* ... */ ... */`). Ob dies allerdings überhaupt notwendig oder sinnvoll ist, sei einmal dahingestellt.

2.2 Literale, Rechenoperatoren, Anweisungen

Literale, also konstante Werte, können Sie in Swift, wie bei anderen Programmiersprachen, einfach in den Code schreiben, wobei die üblichen Regeln gelten: Der Nachkommateil von Gleitkommazahlen wird mit Punkt getrennt, für Strings (Zeichenketten) fungieren doppelte Anführungszeichen als Begrenzer.

```
001 99 // ganze Zahl
002 7.5 // Gleitkommazahl
003 "Hello, playground" // Zeichenkette
```

Zur Darstellung der Grundrechenarten gibt es die aus der Mathematik bekannten Operatoren + für die Addition, – für die Subtraktion, * für die Multiplikation und den Slash (/) für die Division.

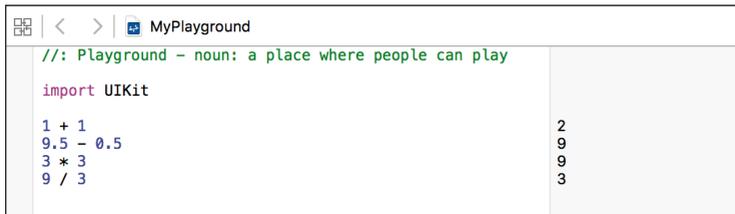


Bild 2.1: Arithmetische Operatoren für die Grundrechenarten

Übrigens handelt es sich bei den im Playground aus Bild 2.1 dargestellten Codezeilen tatsächlich um syntaktisch korrekte Anweisungen, obwohl diese bei realer Ausführung keinerlei Aktionen oder – anders als im Playground – Ausgaben produzieren würden. Swift lässt den Programmierern in dieser Hinsicht genauso wie Objective-C viele Freiheiten. Sogar die Codezeilen

```
001 "Hallo"
002 99
```

mit jeweils nur einem Literal (String sowie ganze Zahl) haben keine Fehlermeldungen oder Warnungen zur Folge, sondern sind syntaktisch gesehen als Anweisungen zu qualifizieren.

Wie Sie sich nach dem Gesagten vermutlich denken können, ist Swift zeilenorientiert. Das heißt, das Ende einer Anweisung geht mit dem Zeilenende einher. Das Setzen eines Semikolons am Ende einer Anweisung ist daher optional. Wenn Sie es aus der Praxis von Objective-C oder anderen Programmiersprachen gewohnt sind, spricht also nichts dagegen, dies zu tun. Es ist auch möglich, mehrere Anweisungen in einer Zeile zu notieren. In diesem Fall müssen Sie die einzelnen Anweisungen – bis auf die letzte – jedoch mit Semikolon abschließen:

```
var x = 12; var y = 7; var z = 1000; print(x + y + z)
```

Die obige Codezeile enthält vier Anweisungen, nämlich drei Variablendefinitionen und eine Ausgabeanweisung. Über deren Bedeutung erfahren Sie im folgenden Abschnitt mehr.

2.3 let und var

Eine Variable ist ein benannter Speicherort, an dem ein Wert abgelegt, gelesen und auch verändert werden kann – »einer«, weil eine Variable jeweils nur einen Wert speichern kann, nicht mehrere gleichzeitig. Von daher hat der Vergleich mit einem Behälter durchaus Sinn.

Wie bei den meisten anderen Programmiersprachen ist auch in Swift einer Variablen immer ein bestimmter Datentyp zugeordnet. Das bedeutet, dass eine einmal definierte Variable grundsätzlich nur Werte eines ganz bestimmten Typs aufnehmen kann (beispielsweise nur Strings oder nur ganze Zahlen).

Swift-Variablen müssen im Code explizit definiert werden, bevor sie verwendet werden können. Dies geschieht mit dem Schlüsselwort `var`. Diesem folgt die Angabe eines Bezeichners, also eines Namens für die Variable.

```
var meineVariable // Definition unvollständig
```

Hier lautet der Name der Variablen `meineVariable`, allerdings ist die Definition noch nicht vollständig. Was noch fehlt, ist die Zuordnung eines Datentyps. Sowohl der Bezeichner einer Variablen als auch deren Datentyp können nach der Definition nicht mehr geändert werden. Es gibt zwei Wege, den Datentyp einer Variablen festzulegen:

- Man notiert einen Doppelpunkt nach dem Variablennamen, gefolgt von einem Leerzeichen und expliziter Angabe des Datentyps.

```
var meineVariable: Int
var deineVariable: String
```

Die erste Codezeile definiert eine Variable zur Aufnahme von ganzen Zahlen, die zweite Codezeile definiert eine String-Variable. Den Variablen können danach in separaten Anweisungen Werte zugewiesen werden, z. B.:

```
meineVariable = 10
deineVariable = "Programmieren mit Swift"
```

Außerdem unterstützt Swift Typinterferenz. Das heißt, der Compiler kann den Datentyp anhand des zugewiesenen Wertes ermitteln. Sie müssen den Datentyp also nicht zwingend – wie in obigem Beispiel – explizit angeben, wenn Sie der Variablen bereits bei der Definition einen Wert zuweisen.

```
var meineVariable = 10
var deineVariable = "Programmieren mit Swift"
```

Die Variable `meineVariable` ist wieder vom Typ `Integer`, also zur Aufnahme von ganzen Zahlen bestimmt, und bei der Variablen `deineVariable` handelt es sich nach der obigen Definition wieder um eine `String`-Variable.

Ausdrücklich sei darauf hingewiesen, dass Folgendes nicht funktioniert:

```
001 var eineVariable // FEHLER
002 // (Es fehlt die Zuordnung eines Datentyps)
003 eineVariable = 17
```

Mit einer separaten Anweisung können Sie die Variable nur initialisieren, wenn Sie bei der Definition das Schlüsselwort für den Datentyp angeben.

```
001 var eineVariable: Int // OK
002 eineVariable = 17
```

Sie können den Datentyp aber auch explizit mit angeben, wenn Sie eine Variable sogleich bei der Definition initialisieren.

```
var eineVariable: Int = 17
```

Das Schlüsselwort für den Datentyp zu notieren kann unter Umständen die Lesbarkeit des Codes erhöhen. Es ist aber in Swift durchaus üblich, die automatische Typerkennung auszunutzen.



EXKURS: DIE ZUWEISUNG

Lassen Sie uns an dieser Stelle für die in der Programmierung weniger versierten Leser kurz auf die Zuweisung eingehen. Dabei handelt es sich um eine spezielle Anweisung, mit deren Ausführung in einer Variablen ein neuer Wert abgelegt wird. Der alte Wert wird dabei gegebenenfalls überschrieben, da eine Variable zur gleichen Zeit ja nur einen einzigen Wert enthalten kann. Bei der ersten Zuweisung an eine Variable spricht man von Initialisierung: Die Variable wird mit einem Wert initialisiert.

Die Zuweisung wird mit dem aus der Mathematik bekannten Gleichheitszeichen (dem Zuweisungsoperator) dargestellt, das hierbei aber eine ganz andere Bedeutung hat. Eine Zuweisung setzt sich insgesamt aus drei Teilen zusammen:

- dem Zuweisungsoperator (=)
- einem linken Operanden; er besteht immer aus genau einem Variablennamen, wobei man vom Bezeichner der Variablen spricht. Damit wird die Variable bezeichnet, die mit der Zuweisung einen neuen Wert erhält.
- dem rechten Operanden, der den zuzuweisenden Wert darstellt. Hier darf auch ein komplexer Ausdruck stehen, der nach seiner Berechnung den zuzuweisenden Wert ergibt.

Hier ein paar Beispiele:

```
var zahl = 33 // Definition der Variablen zahl
zahl = 22
```

Die Variable `zahl` wird gleich bei der Definition mit dem Anfangswert 33 versehen (initialisiert). Nach der Zuweisung `x = 22` besitzt die Variable den Wert 22.

```
zahl = 8 * 3 - 11
```

Nach Ausführung dieser Anweisung enthält die Variable `zahl` den Wert 13. Daraus wird auch deutlich, dass eine Zuweisung immer von rechts nach links abgearbeitet wird:

- 1 Zunächst wird der Ausdruck auf der rechten Seite ausgewertet.
- 2 Anschließend wird das Ergebnis der auf der linken Seite bezeichneten Variablen zugewiesen.

```
zahl = zahl + 7
```



Nach Ausführung der obigen Anweisung enthält `zahl` nunmehr den Wert 20, nämlich 13 plus 7. Es gilt:

- Wenn der Bezeichner einer Variablen auf der rechten Seite einer Zuweisung steht, wird der aktuelle Wert der Variablen gelesen.
- Auf der linken Seite der Zuweisung bezeichnet der Variablenname die Speicherstelle, »die etwas bekommt«, nämlich den Wert, der sich nach Auswertung des Ausdrucks auf der rechten Seite ergibt.

```
zahl = "Swift" // FEHLER
```

Die obige Anweisung ist fehlerhaft, da im Zuge der Definition durch die Zuweisung des Wertes 33 an die Variable `zahl` für diese per Typinterferenz der Datentyp `Integer` festgelegt wurde und einer `Integer`-Variablen kein `String` zugewiesen werden kann.

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

import UIKit

var zahl = 33           33
zahl = 22              22
zahl = 8 * 3 - 11     13
zahl = zahl + 7       20
zahl = "Swift"        ❗ Cannot assign a value of type 'String' to a value of type 'Int!'
```

Bild 2.2: Neben fehlerhaften Codezeilen erscheint im Playground ein Symbol mit weißem Ausrufezeichen auf rotem Hintergrund. Klicken Sie das Symbol gegebenenfalls an, um die komplette Fehlermeldung zu sehen.

Eine App arbeitet häufig mit unveränderlichen Werten, wie z. B. dem aktuellen Mehrwertsteuersatz, der maximalen Anzahl von Login-Versuchen, die Sie als Programmator erlauben wollen, oder bei Spielen mit einer maximal zu erreichenden Punktzahl. Dabei ist es wenig praktikabel, die entsprechenden Werte als Literale in den Code zu schreiben, vor allem dann nicht, wenn sie öfter vorkommen. Wenn sich z. B. der Mehrwertsteuersatz ändert, müssten Sie den Wert an jeder Stelle, an der er auftritt, austauschen. Verwenden Sie dagegen Konstanten, müssen Sie den Code nur an einer einzigen Stelle ändern. Außerdem erhöht sich die Lesbarkeit des Quellcodes, da eine Konstante wie `mwst`, `anzahlErlaubterLogins` oder `maxPunkte` natürlich aussagekräftiger ist als z. B. die Literale 19, 3 oder 100 (falls man

drei Login-Versuche erlauben möchte bzw. in einem Spiel die maximal zu erreichende Punktzahl 100 beträgt).

Wie bei Variablen handelt es sich auch bei Konstanten um einen mit Namen ansprechbaren Speicherort – von daher spricht man auch von »benannten Konstanten«, um sie von Literalen zu unterscheiden –, und Konstanten müssen ebenfalls vor ihrer Verwendung definiert werden. Dies geschieht mit dem Schlüsselwort `let`.

```
let maximum = 100
```

Während Variablen ihren Wert durch Neuzeuweisung ändern können, ist das bei Konstanten – wie der Name schon sagt – nicht der Fall.

```
001 let maximum = 100
002 maximum = 150 // FEHLER
```

Die obige Neuzeuweisung ist fehlerhaft, da `maximum` als Konstante definiert wurde. Die Verwendung von Konstanten für Werte, die während der App-Ausführung unveränderlich bleiben sollen, dient somit dem Schutz des Programmierers vor eigenen Fehlern. Gegenüber Variablen haben Konstanten zudem den Vorteil, dass sie effizienter verwaltet werden (der Compiler »weiß«, dass sich der Wert nicht mehr ändern wird) und somit ressourcensparender sind.

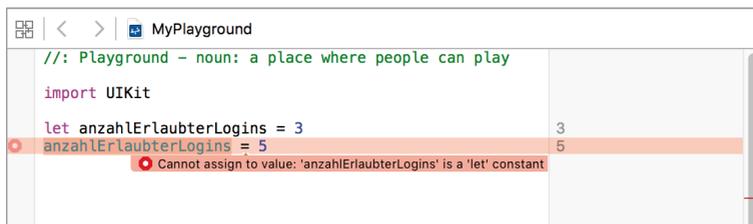


Bild 2.3: Die erneute Zuweisung eines Wertes an eine Konstante macht sich auch im Playground als Fehler bemerkbar.



INFO:

Sie müssen den Wert einer Konstanten jedoch nicht zwangsläufig bei der Definition zuweisen, sondern können dies wie bei Variablen auch in einer separaten Anweisung tun.

```
let anzahlErlaubterLogins: Int
anzahlErlaubterLogins = 3 // OK
```

Nach der ersten Zuweisung dürfen Sie jedoch keine weitere Zuweisung durchführen.

Wenn Sie mehrere Variablen oder Konstanten benötigen, können Sie sie auch mit einer einzigen Anweisung definieren, indem Sie die einzelnen Ausdrücke durch Kommas trennen:

```
var zahl1 = 0, zahl2 = 0, zahl3 = 0
```

oder

```
var zahl1: Int, zahl2: Int, zahl3: Int
```

Die erste Anweisung definiert die Integer-Variablen `zahl1`, `zahl2`, `zahl3` und initialisiert sie mit dem Wert 0. Die zweite Anweisung definiert die Variablen ebenfalls mit dem Datentyp `Integer`, ohne sie jedoch zu initialisieren. Die beiden Anweisungen sind natürlich alternativ zu verstehen, da die Bezeichner von Variablen eindeutig sein müssen. Wenn Sie Konstanten benötigen, verwenden Sie das Schlüsselwort `let`:

```
let anzahlErlaubterLogins = 3, maxPunkte = 100
```

Die obigen Beispiele definieren durchweg Variablen bzw. Konstanten gleichen Datentyps. Eine Einschränkung in dieser Hinsicht besteht jedoch nicht. Sie können in einer Codezeile auch Variablen bzw. Konstanten unterschiedlichen Datentyps definieren:

```
var anzahlLogins = 0, str = "Welcome to Swift"
```

2.4 Bezeichnerwahl

Für verschiedene Elemente der Programmiersprache, wie z. B. Variablen, Konstanten, Klassen oder Funktionen, vergeben Sie Namen, wenn Sie sie im Code definieren. Der Fachausdruck für solche Namen lautet Bezeichner.

Grundsätzlich dürfen Sie Bezeichner unter Beachtung einiger Regeln frei wählen:

- Das erste Zeichen darf keine Ziffer sein.
- Swift ist case-sensitiv, das heißt, es unterscheidet zwischen Groß- und Kleinschreibung. Die Bezeichner `swiftIsCool` und `swiftiscool` sind daher nicht identisch, sondern bezeichnen im Code zwei unterschiedliche Elemente.
- Bezeichner müssen eindeutig sein. Dies hat zur Folge, dass innerhalb eines Gültigkeitsbereichs nicht der gleiche Bezeichner für zwei verschiedene Elemente vergeben werden darf.
- Ebenso wenig dürfen Bezeichner mit Schlüsselwörtern gleichlauten, die ja bereits eine in der Programmiersprache Swift festgelegte Bedeutung haben. Es gibt jedoch eine Ausnahme: Wenn Sie das Schlüsselwort mit den Zeichen ``...`` (oder `´...´`) begrenzen, können Sie den Namen als Bezeichner verwenden. Der Aufwand erscheint allerdings unbegründet, und eine Notwendigkeit, dies zu tun, ist auch nicht ersichtlich.

Ansonsten dürfen Swift-Bezeichner nahezu alle Zeichen des Unicode-Zeichensatzes enthalten, also auch deutsche Umlaute und sogar chinesische, japanische, griechische Schriftzeichen etc. Für die Kreiszahl Pi könnten Sie z. B. auch eine Konstante mit dem Bezeichner π definieren. Einige Zeichen sind jedoch in Bezeichnern nicht erlaubt. Dazu gehören, wie in jeder anderen Programmiersprache, Leerzeichen (der Compiler könnte ansonsten nicht ermitteln, wo ein Bezeichner endet), aber auch verschiedene Sonderzeichen wie z. B. mathematische Symbole, Punkte, Pfeile oder Rahmenzeichen. Am besten fahren Sie, wenn Sie auf Sonderzeichen gänzlich verzichten. Unterstriche in Bezeichnern sind zwar erlaubt, jedoch in der Swift-Programmierung nicht üblich.



UNICODE

Der Unicode-Zeichensatz ermöglicht die Darstellung von bis zu 65536 verschiedenen Zeichen. Er enthält alle Sonderzeichen und sogar jedes Schriftzeichen jeder beliebigen Landessprache. Jedem Zeichen ist eine Ordnungszahl eindeutig zugeordnet. Die Ordnungsnummern sind fortlaufend. Die ersten 256 Zeichen entsprechen dabei der früher durchweg verwendeten ASCII-Tabelle.

Für die Bezeichner von Variablen und Konstanten wird in Swift die *lowerCamelCase*-Schreibweise empfohlen. Nach dieser beginnt jeder Bezeichner mit einem Kleinbuchstaben. Setzt sich ein Bezeichner aus mehreren Wörtern zusammen, dann wird jedes weitere Wort großgeschrieben. Hier ein paar Beispiele: `meinMacBook`, `bigPoint`, `teilnehmer` (setzt sich nicht aus mehreren Wörtern zusammen), `anzahlTeilnehmer`, `arrayOfSelectedItem`, `minValue`, `maxValue`.

Hier ein paar weitere Beispiele für gültige Bezeichner:

```
001 var kmProSek = 0
002 let  $\pi$  = 3.14159265359
003 var 量 = "biànliàng"
004 var myVar1 = 99
005 var myVar2 = 5000
```

Folgende Bezeichner sind dagegen nicht korrekt:

```
001 var my Var: Int // keine Leerzeichen in Bezeichnern
002 let 1A = 100 // das erste Zeichen eines Bezeichners
003 // darf keine Ziffer sein
004 var Int = "Integer" // Int ist ein Schlüsselwort zur
005 // Bezeichnung des Datentyps
006 // und darf deshalb nicht zur
007 // Bezeichnerwahl herangezogen werden
```

Und eine weitere Empfehlung: Grundsätzlich können Sie für die Bezeichner von Variablen oder Konstanten auch nur einen einzigen Buchstaben vorsehen, z. B. `x`, `y` oder `z`, und in einigen Fällen ist das auch praktikabel – etwa bei einer Schleifenvariablen, die ansonsten keine spezielle Bedeutung hat. Im Allgemeinen sollten Sie jedoch »sprechende Namen« bevorzugen. Das macht den Quellcode besser lesbar. Bei einer Variablen `skonto` weiß man beispielsweise sofort, was gemeint ist, während die Bedeutung einer Variablen `x` oder `s` (für Skonto) allenfalls aus dem Zusammenhang erkennbar ist.

Ein ganz besonderes Gimmick der Programmiersprache Swift sei in diesem Zusammenhang noch erwähnt: Swift erlaubt es, als Bezeichner sogenannte Emoji-Symbole zu verwenden. Emoji-Symbole stammen ursprünglich aus Japan. Sie werden dort seit längerem vergleichbar den bei uns geläufigen Emoticons eingesetzt.

INFO:

Emoticons sind Symbole, die Stimmungen wiedergeben. Die bei uns am meisten verwendeten Emoticons sind

```
: - )
: - (
und
; - )
```



Zwar haben die japanischen Mobilfunkbetreiber eigene Standards in Bezug auf Tastatur und Zeichenbelegung. Infolge der häufigen Verwendung und auch auf Betreiben von Apple und Google entschied sich das Unicode-Konsortium jedoch, 600 Emoji-Zeichen in den Unicode-Zeichensatz aufzunehmen. Ob sich der Aufwand bei der Programmierung für Sie lohnt, sollten Sie für sich selbst entscheiden.

```
let 🐶 = "Mein Hund"
print(🐶)
```

Bild 2.4: Emoji-Zeichen als Bezeichner einer Konstanten

TIPP:

Emojis sind als Nicht-Buchstabenzeichen mit den deutschen Tastaturen auf einfache Weise nicht darstellbar. Xcode stellt jedoch eine Vielzahl von Emojis, geordnet nach Kategorien (*Personen, Natur, Essen & Trinken, Feier, Aktivität, Reisen & Orte, Objekte & Symbole* usw.), im Menü zur Verfügung. Wählen Sie in der oberen Menüleiste *Edit/Emoji & Symbols*. Im erscheinenden Fenster brauchen Sie ein Emoji nur auszuwählen, um es im Code zu verwenden. Es wird dann an der aktuellen Cursorposition eingefügt. Per Eingabe in das Suchfeld können Sie auch gezielt nach bestimmten Emojis suchen. Beachten Sie jedoch, dass Sie englische Suchbegriffe verwenden müssen. Suchen Sie beispielsweise nicht nach »Hund«, sondern nach »dog«.



STICHWORTVERZEICHNIS

Symbole

--, Dekrementoperator 50
-=: Subtraktion und Zuweisung 51
!, logische Negation 55
 Unwrap-Operator 60, 93
!=, Vergleichsoperator 53
?, Bedingungsoperator 68
??: nil coalescing operator 64
..., Bereichsoperator 52
..<=: Bereichsoperator 52
*/=: mehrzeiliger Kommentar, Ende 21
*=: Multiplikation und Zuweisung 51
/*=: mehrzeiliger Kommentar, Beginn 21
//=: einzeiliger Kommentar 20
/=, Division und Zuweisung 51
&&, logisches Und 55
%, Modulo-Operator 48
%=, Modulo und Zuweisung 51
++, Inkrementoperator 50
+=: Addition und Zuweisung 51
<=: Vergleichsoperator 53
==, Vergleichsoperator 53
>=: Vergleichsoperator 53
>=, Vergleichsoperator 53
||, logisches Oder 55
2-Tupel 72
\n, Steuerzeichen 38

A

Action 17
Addition 22
Aktualisierung, for-Schleife 76
Aktualisierungsteil, for-Schleife 77
Anweisung 22
append, Methode von Arrays 88
Apple 83
Apple-Developer-Programm 10
Apple Watch 9
App Store 8, 10
arc4random, Funktion 74
arc4random_uniform, Funktion 74
Argumente 104

Array

Definition 82
durchlaufen 85
Elemente überschreiben 91
Indexgrenzen 85
Index und Wert von Elementen extrahieren 101
initialisieren 83
leeres, definieren 83
neu erstellen 90
zuweisen 98
Array-Bereiche, ersetzen 91
Array<Int> 82
Array-Literal 83, 88, 90
 leeres 90
Array-Operationen 87
Arrays 82
ASCII 30
Assistant Editor, Playground 39
Assoziativität
 von logischen Operatoren 58
 von Operatoren 58
 von Vergleichsoperatoren 58
Attributes inspector 13, 14
Ausdrücke
 boolesche 54
 logische 54
Ausgabekonzole, im Playground 62
Ausgaben
 in der Benutzeroberfläche 61

B

Background, Listenfeld 14
Backslash 39
Basisklasse 153
Bedingung, for-Schleife 76
Bedingungsoperator 68
Bereichsoperator
 geschlossener 52
 halboffener 53
Bereichsoperatoren 48, 52, 91
Bezeichner 23, 29, 81
 von Enumerationen 134
 von Strukturen und Klassen 134
Bezeichnerwahl 16, 29

Bool

Schlüsselwort 37
Schlüsselwort zur Bezeichnung des Datentyps 32
break-Anweisung 79
break-Befehl 70
Breakpoint 12
Button
 Steuerelement 13, 17

C

C# 154
capacity, Eigenschaft von Arrays 89
case, Schlüsselwort, Enumeration 142
case-Marke, switch-Anweisung 69
case-Zweig, switch-Anweisung 69
Casting, von Strings 46
Character, Datentyp 41
characters, Eigenschaft 86
Character-Variable 41
Character-Werte 41
class, Schlüsselwort 141, 150
Closure-Ausdrücke 120, 127
Closure 120, 127
 definieren 127
 Kurzschreibweisen 132
Cocoa, Framework 19
Compiler 21, 139, 142
Computed Properties 146, 147
Computed Property
 als Read-Only-Eigenschaft 149
 Definition 148
 geerbte, überschreiben 154
 statische 150
Connect 17
Connection 16
console output, Playground 39
continue, Anweisung 80
count
 Eigenschaft von Arrays 87
 Eigenschaft von Dictionaries 94

D

Datenstruktur 82

- Datentyp
 - boolescher 32, 37
 - definieren 135
 - komplexer 82
 - von Funktionen 120
- Datentypen 20, 23
 - elementare 32, 59, 138
- DBL_MAX, Konstante 36
- DBL_MIN, Konstante 36
- Debug-Abschnitt, Xcode 12
- Debug-Bereich, Playground 39
- Debuggen 12
- default-Marke
 - switch-Anweisung 69
- Dekrementoperator 48, 50
 - Postfix-Notation 50
 - Präfix-Notation 50
- Developer Tools 9
- Dictionary 82, 91, 93
 - Definition 92
 - leeres, erstellen 92
 - Schlüssel und Wert von
 - Elementen extrahieren 101
 - zuweisen 98
- Dictionary-Element
 - entfernen 97
 - Wert entpacken 93
- Division 22
- Double
 - Datentyp 32, 35
 - Datentyp, Genauigkeit 36
 - Schlüsselwort zur Bezeichnung des Datentyps 32
- E**
- Eigenschaft
 - initialisieren 146
 - statische 135, 150, 151
 - von Objekten, initialisieren 137
- Eigenschaften 42, 135, 147
 - von Arrays 87
- Einfachvererbung 154
- Eingaben, in der
 - Benutzeroberfläche 61
- else, Schlüsselwort 67
- else-if-Zweige 67
- else-Zweig 67
- Emojis 31
- Emoji-Symbole
 - als Bezeichner verwenden 31
- Emoticons 31
- Entwickler-Tools 8
- Entwicklungsumgebung
 - integrierte 8
- enum, Schlüsselwort 142
- Enumeration
 - Kurzschreibweise 143
- Enumerations 134, 142
- Ereignisbehandlungsmethoden 17
- Execute Playground 75
- F**
- fallthrough 70
- false, Wahrheitswert 37, 54
- final 155
- Finder 9
- Float
 - Datentyp 32, 35
 - Datentyp, Genauigkeit 36
 - Schlüsselwort zur Bezeichnung des Datentyps 32
- for-in-Schleife 66
 - auf Bereiche anwenden 86
 - Schleifenvariable 85
 - Strings durchlaufen 86
- forKey 96
- for-Schleife 76, 77
- func, Schlüsselwort 102
- Funktion 16, 102
 - als Parameter für andere Funktionen 122
 - als Rückgabewert 124
 - anonyme 127
 - aufrufen 102
 - definieren 102
 - mit mehreren Rückgabewerten 108
 - verschachteln 124
- Funktionsaufruf 104
- Funktionsausdruck 127
- Funktionsdefinition 104
- Funktionskopf 102
- Funktionsrumpf 102
- Funktionstypen 120, 121
- G**
- get, Methode 148
- Getter 147, 148
- Gleitkommawert, Begriff 36
- Gleitkommazahl 21
 - Begriff 36
- Gleitkommazahlen 35
- H**
- Hauptbereich, Xcode 12
- Hintergrundfarbe
 - der App, einstellen 14
- I**
- IBAction 14
- IBOutlet 14
- IDE 8
- if, Schlüsselwort 66
- if-Anweisung 60, 61, 66
- if-Bedingung 66
- if-let-Anweisung 18, 47, 63, 93
- implicit unwrapped optional 63
- Import-Anweisung 19, 74, 152
- init
 - Methode 138, 139, 142
 - Methode, in Enumerationen 145
 - Methode, von elementaren Datentypen 138
- Initialisierer 58
- Initialisierung
 - for-Schleife 76
 - von Variablen 25
- Initialisierungsteil, for-Schleife 77
- Initializer 58, 138
 - parameterloser 142
- Inkrementoperator 48, 50
 - Postfix-Notation 50
 - Präfix-Notation 50
- inout
 - Parameter 118
 - Schlüsselwort 118
- In-Out-Parameter 119
- Instanzen 135
 - aktuelle 138
 - erstellen 135

- Int
 - Datentyp 32
 - Schlüsselwort zur Bezeichnung des Datentyps 32
 - Int8, Datentyp 33
 - Int16, Datentyp 33
 - Int32, Datentyp 33
 - Int64, Datentyp 33
 - Integer 24
 - Datentyp 26, 28, 32, 49
 - Integer-Datentypen 32
 - Integrated Development Environment 8
 - Interface Builder 61
 - internal, Zugriffsmodifizierer 152
 - iOS App Store 10
 - iOS-Playground 19
 - iOS-Projekte 10
 - iPad 9, 10
 - iPhone 9, 10
 - isEmpty
 - Eigenschaft 42
 - Eigenschaft von Arrays 87
 - Eigenschaft von Dictionaries 95
- J**
- Java 154
- K**
- keepCapacity 89, 98
 - keys
 - Array 94
 - Eigenschaft von Dictionaries 94
 - Klasse 134, 141, 146
 - abgeleitete 153, 154
 - Definition 141
 - finale 155
 - Vererbung 153
 - Klassendefinition 135, 136
 - Kommentare 20
 - einzeilige 20
 - mehrzeilige 20
 - verschachteln 21
 - Kommentarzeichen 21
 - Konkatenation 40
 - Konstanten 20, 27
 - initialisieren 58
 - von Enumerationen, Werte zuweisen 144
 - Konstruktor 59, 138
 - Konstruktormethode 138
 - Kontrollbereich
 - aus-/einblenden, Xcode 12
 - Xcode 12, 14
 - Kontrollstrukturen 66, 67
 - verschachteln 67
 - Konvertierung 44
 - explizite 45
 - implizite 44
 - von Gleitkommawerten 46
 - von Strings 46
 - Koordinatensystem
 - zweidimensionales 72
 - Kurzschreibweisen
 - für Closures 131
 - für Zuweisungsoperatoren 51
- L**
- Label, Steuerelement 13, 16, 61
 - Labeled Statement 80
 - Lambda-Ausdruck 127
 - Laufvariable, for-Schleife 77
 - Launchpad 9
 - lazy, Schlüsselwort 146
 - Lazy Properties 146
 - leere Zeichenkette 38
 - let
 - Schlüsselwort 27
 - Schlüsselwort, Dictionary 95
 - Listen, durchlaufen 85
 - Literal 21
 - Literale, zur Darstellung von
 - Gleitkommazahlen 35
 - lowerCamelCase-Schreibweise 30
- M**
- Mac App Store 8, 10
 - Mac-Tastatur
 - geschweifte Klammer darstellen 66
 - Pipe-Zeichen darstellen 55
 - Main.storyboard
 - Datei 13
 - max, Eigenschaft 34, 42
 - Methode
 - finale 155
 - geerbte, überschreiben 154
 - öffentliche 152
 - statische 150, 151
 - Methoden 16, 42, 135
 - von Arrays 87
 - min, Eigenschaft 34, 42
 - Module 152
 - Modulo-Operation 48
 - mit Gleitkommawerten 49
 - Modulo-Operator 48
 - Multiplikation 22
 - mutating, Schlüsselwort 137
 - Mutating-Methode 137
- N**
- Navigationsbereich
 - aus-/einblenden 12
 - Xcode 12
 - Negation, logische 55, 56
 - Newline-Zeichen 38
 - nil 97
 - Schlüsselwort 59
 - nil coalescing operator 64
 - Nullzeiger 59
- O**
- Objective-C 10, 59, 69, 70, 99, 154
 - Objekt, erstellen 135
 - Objekte 135
 - Oder, logisches 55, 56
 - OOP 134
 - OOP-Konzepte
 - spezielle 146
 - Operationen, mit Arrays 87
 - Operatoren
 - arithmetische 22, 48
 - binäre 48, 54
 - logische 48, 54
 - logische, Assoziativität 58
 - logische, Priorität 57
 - spezielle 48
 - ternäre 68
 - unäre 48
 - Optional 46, 47
 - automatisch entpacken 63
 - entpacken 60, 63

Optionals 18, 59, 60, 96
 OS-X-Playground 19
 OS-X-Projekte 10
 Outlet 16, 17
 override 154

P

Parameter 102, 104
 aktuelle 104
 benannte 35, 103, 110
 formale 104
 Standardwert festlegen 112
 unbenannte 110
 von Funktionen 103
 Parameterliste 104
 Pipe-Zeichen 55
 Playground 10, 18, 20
 Ausgabekonsole 39, 62
 Code ausführen 75
 Debug-Bereich 39
 Ein- und Ausgabeszenarios
 simulieren 61
 erstellen 10
 Quick Look 42
 Postfix-Notation
 Dekrementoperator 50
 Inkrementoperator 50
 Präfix-Notation
 Dekrementoperator 50
 Inkrementoperator 50
 print, Funktion 34, 35, 102
 Priorität
 von logischen Operatoren 57
 von Vergleichsoperatoren 57
 private, Zugriffsmodifizierer 146,
 152, 153
 Programmierung
 funktionale 134
 objektorientierte 134, 146
 Programmverzweigungen 66
 Project Builder 9
 Projekte, erstellen 10
 Projekteinstellungen 10, 12
 Projektnavigator, Xcode 12
 Projektvorlage 10
 Properties 135
 public, Zugriffsmodifizierer 152

Q

Quellcode 21
 Quick Look, Playground 42, 72

R

rawValue, Eigenschaft von
 Enumerationen 144
 Read-Eval-Print-Loop 19
 Read-Only-Eigenschaft 149
 Kurzschreibweise 149
 Referenztypen 141
 removeAll(keepCapacity)
 Methode von Arrays 89
 Methode von Dictionaries 98
 removeAtIndex
 Methode von Arrays 89
 removeLast
 Methode von Arrays 89
 removeValueForKey
 Methode von Dictionaries 97
 repeat-while-Schleife 75
 REPL 19
 reserveCapacity
 Methode von Arrays 89
 return 105
 Schlüsselwort 124
 return-Anweisung 106
 Richtungsvektoren 72
 Rückgabewert,
 von Funktionen 105

S

Schleifen 66, 73
 mit Sprungmarke versehen 80
 Schleifenbedingung 74
 for-Schleife 76
 Schleifenvariable
 for-in-Schleife 85
 for-Schleife 77, 78
 Schlüssel, von Dictionaries 92
 Schlüssel-Wert-Paar 82, 92
 Schrägstrich,
 umgekehrter, darstellen 39
 self, Schlüsselwort 138
 set, Methode 148
 Setter 147, 148
 Show Result
 Playground 42
 Simulator 13, 18
 Zielgerät ändern 12
 Sprunganweisungen 80
 Sprungmarke 80
 Bezeichner 81
 Standardparameter 113, 115
 Standardwert, für Parameter
 festlegen 112
 static, Schlüsselwort 150
 Steuerelemente
 Eigenschaften anpassen 13
 Steuerzeichen 38
 Stored Property 147
 statische 150
 Storyboard 12, 13, 15
 String 21, 23, 38
 Datentyp 32, 38
 Schlüsselwort zur Bezeichnung
 des Datentyps 32
 Werte einbauen 40
 String-Interpolation 40
 String-Konkatenation 40
 String-Literale 38
 Stringverknüpfung 48
 struct, Schlüsselwort 134
 Struktur 42, 134
 Definition 134
 Mutating-Methode 137
 Strukturdefinition 135, 136
 Subtraktion 22
 Swift, Typsicherheit 32
 Swift 1 111, 150
 println()-Funktion 35
 Swift 2 9, 86, 103, 111, 113, 150
 Swift-Compiler 21, 107
 switch, Schlüsselwort 69
 switch-Anweisung 69
 Bereichsprüfungen 71
 mit Sprungmarke versehen 80
 Tupel vergleichen 72
 Vergleichswerte 69
 switch-Ausdruck 69

T

text-Eigenschaft
 Label-Steuerelement 14, 18
 Touch Up Inside, Ereignis 17
 Trailing Closures 131

- true, Wahrheitswert 37, 54
- Tupel 71, 82, 99
 - auslesen 100
 - erstellen 99
 - in switch-Anweisung vergleichen 71
- Tuple. Siehe Tupel
- Type Methods 150
- Type Properties 150
 - initialisieren 150
- Typerkennung, automatische 24
- Typinterferenz 24, 26, 32, 33, 83
 - bei Funktionstypen 121
- Typkonvertierung 44
 - explizite 45
 - implizite 44
- Typsicherheit 32
- Typüberprüfung 105
- Typumwandlungen 44
 - explizite 45
 - implizite 44
- U**
- UIButton 17
- UIKit, Framework 19, 61
- UILabel 16, 61
- UInt, Datentyp 33
- UInt8, Datentyp 33
- UInt16, Datentyp 33
- UInt32, Datentyp 33
- UInt64, Datentyp 33
- UITextField 61
- Und, logisches 55
- Unicode 29
- Unicode-Zeichensatz 30
- Unsigned 33
- Unsigned-Integer-Datentypen 33
- Unterstrich
 - als Tausender-Trennzeichen 71
- Unwrap-Operator 60, 93
- updateValue(forKey)
 - Methode von Dictionaries 96
- Utilities, Xcode 13
- V**
- Value Binding 72, 73
- values
 - Array 94
 - Eigenschaft von Dictionaries 94
- var
 - Schlüsselwort 23, 116
 - Schlüsselwort, Dictionary 95
 - Schlüsselwort, Tupel 100
- Variable 16, 20, 23, 25
 - boolesche 32
 - dekrementieren 50
 - Funktionstyp zuweisen 122
 - initialisieren 58
 - inkrementieren 50
 - optionale 59
 - optionale, initialisieren 59
- Variadic-Parameter 114, 115
- Vererbung 146, 153
- Vererbungsbeziehung,
 - einrichten 153
- Vergleichsoperator 48, 53
 - Assoziativität 58
 - Priorität 57
- Verzweigungsanweisungen 73
- Verzweigungskonstrukte 66
- ViewController 15
 - Klasse 16
- ViewController.Swift
 - Datei 15, 16
- viewDidLoad 15
- W**
- Wahrheitswerte 32
- watchOS 2 9
- Wert
 - boolescher 54
 - konstanter 21
 - optionaler 60
 - optionaler, automatisch
 - entpacken 63
 - optionaler, Dictionary 93
- Werttypen 141
- where-Klausel 73
- while, Schlüsselwort 74
- while-Schleife 73, 74
- Wiederholungsanweisungen 73
- Wrapper 60
- X**
- Xcode 8
 - Debug-Bereich einblenden 39
 - Emojis darstellen 31
 - Projekt anlegen 9
 - Swift-1-Projekte konvertieren 35
- Xcode 1.0 8
- Xcode 7
 - Playground-Ausgabekonsole 39
- Xcode-Entwickler-Tools 8
- Z**
- Zahlen,
 - ohne Nachkommastellen 32
- Zählschleife 76, 77
- Zeichenkette 21, 38
 - leere 38
- Zeichensatz 30
- Zufallszahlen 74
- Zugriffsmodifizierer 152
- Zugriffsmodus, auf Elemente einer Klasse oder Struktur 152
- Zuweisung 25
- Zuweisungsoperator 25, 48
 - zusammengesetzter 96

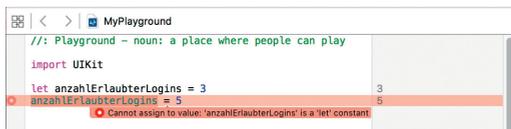


SCHNELLEINSTIEG SWIFT 2

Egal ob Sie bereits für Apple iOS Apps entwickeln oder damit anfangen wollen: um die Programmiersprache Swift kommen Sie nicht herum. Apple hat alles über Bord geworfen und von Grund auf eine neue Sprache entwickelt. Endlich, werden viele Objective-C-Entwickler sagen. Und Neulinge brauchen sich nicht mehr mit den Altlasten auseinanderzusetzen. Nun liegt bereits die zweite Generation der Sprache Swift vor, und Entwickler müssen sich mit den neuen Konzepten auseinandersetzen. Dieses Buch zeigt Ihnen, worauf es dabei ankommt.



Die Nutzung der Entwicklungsumgebung Xcode wird anhand von Bildern erklärt.



Typische Fehler während der Programmierung werden erklärt, damit Sie sie vermeiden.

In neun Schritten zum Erfolg

Nach der Installation der Entwicklungsumgebung Xcode steigt der Autor direkt in die Swift-Programmierung ein, angefangen bei der grundlegenden Syntax wie den Kommentaren, Literalen, Rechenoperatoren, Anweisungen und Datentypen bis zu den Kontrollstrukturen. Damit können Sie direkt erste Programme mit Swift umsetzen. Für die ersten Gehversuche müssen Sie kein großes Xcode-Projekt beginnen, sondern können sich mit dem Playground in der IDE begnügen.

Danach geht es direkt weiter mit Funktionen, Funktionstypen und Closure-Ausdrücken. Die letzten beiden Kapitel im Buch beschäftigen sich mit der objektorientierten Programmierung. Sie lernen dabei auch OOP-Konzepte wie Lazy und Computed Properties kennen.

Aus dem Inhalt:

- Xcode einrichten und verwenden
- Kommentare, Literale, Rechenoperatoren
- Anweisungen und elementare Datentypen
- Implizite und explizite Typkonvertierungen
- Operatoren, Initializer und Optionals
- Kontrollstrukturen
- Arrays, Dictionaries und Tupel
- Funktionen in Swift
- Funktionstypen und Closure-Ausdrücke
- Mit Swift objektorientiert programmieren
- Spezielle OOP-Konzepte



9 783645 602563

Besuchen Sie
unsere Website
www.franzis.de

FRANZIS