

Dokumentation der Demo

Swift

Version 1.0

Diese Dokumentation dient als kleiner Orientierungsleitfaden für die in class Demo. Diese ist vor allem für diejenigen, die am Tag meines E-Portfolios nicht anwesend sind und die Inhalte der Demo nacharbeiten möchten. Alle anderen erwartet diese Demo vorgeführt und ausführlich erklärt in meinem E-Portfolio. Sollten Fragen bei der Anmeldung oder bei der Nacharbeit der Demo kommen, einfach eine E-Mail an: kathy11997@gmx.de

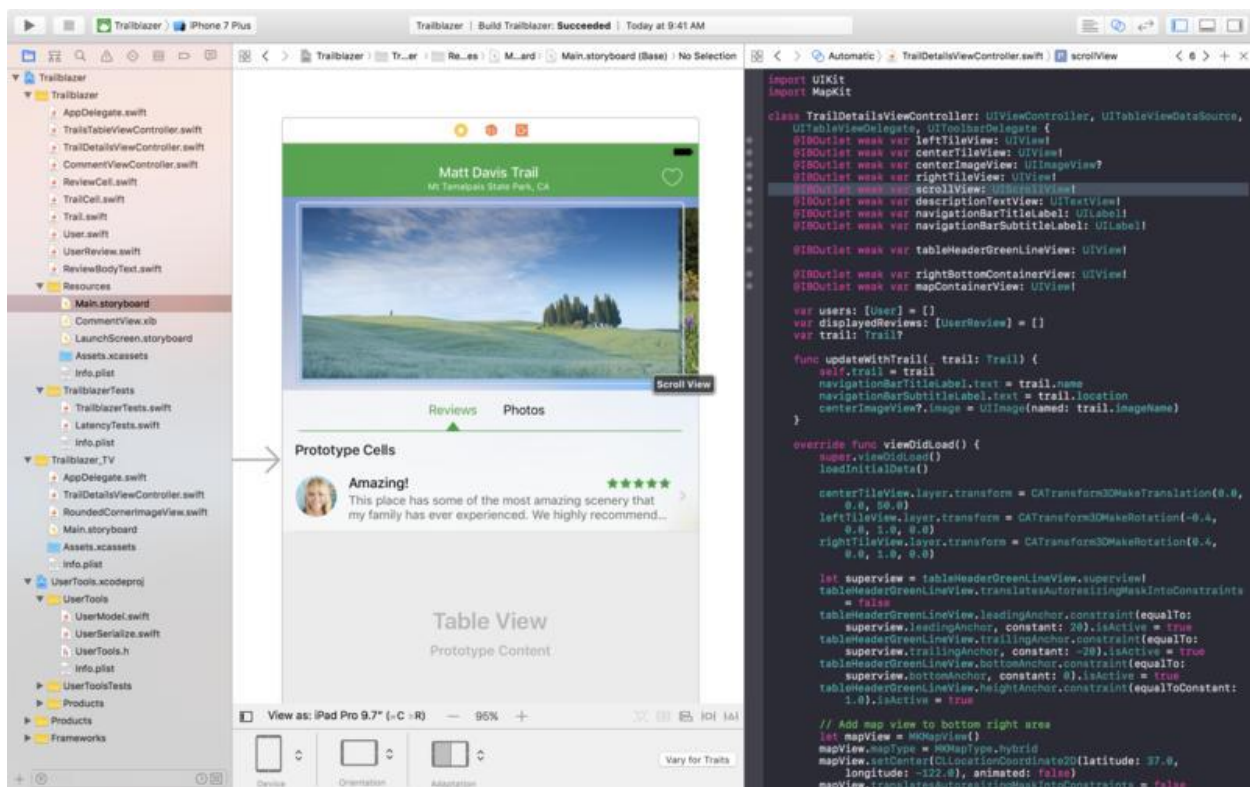
Autor des Dokuments	Katharina-Maria Heer	Erstellt am	09.06.2017
Dateiname	Dokumentation_Live_Demo		
Seitenanzahl	7	© 2017 Katharina-Maria Heer	

Tutorial für die Demo: Schritt für Schritt

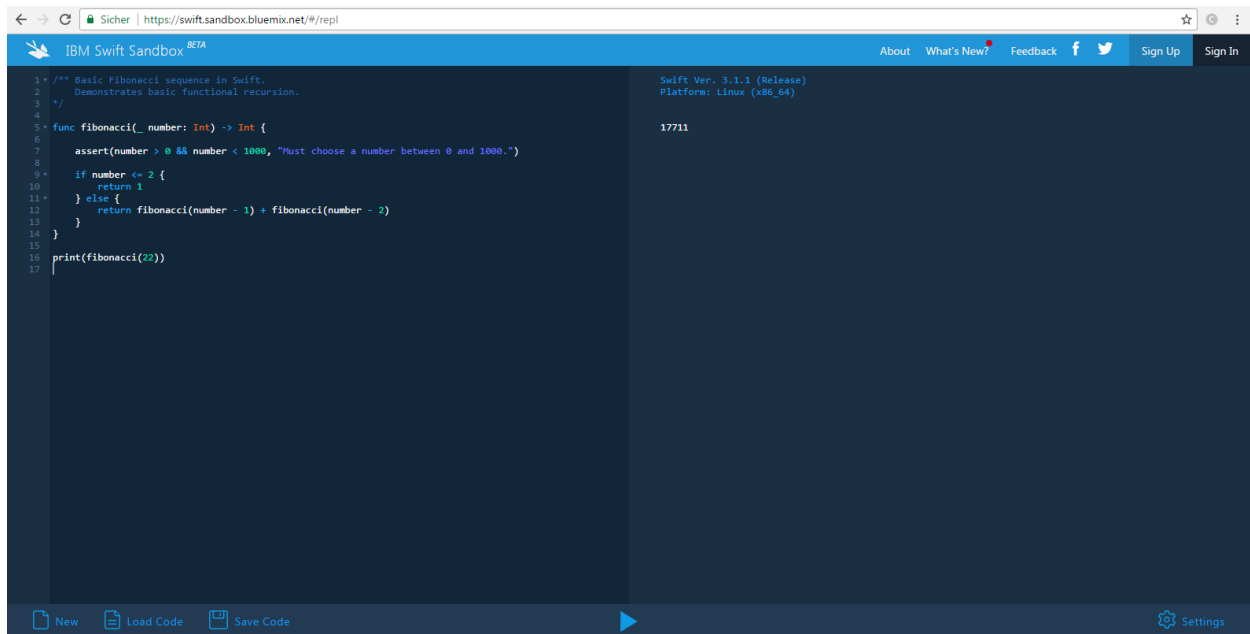
Die Demo wurde entwickelt um die Grundlagen von Swift näher kennenzulernen. Es werden also die Grundstrukturen anhand von Codesnippets erklärt, sodass der Nutzer auf Grundlage dieses Wissens schon einige kleine Swift Programme entwickeln kann.

Entwicklungsumgebung

Um mit Swift arbeiten zu können ist eine geeignete Entwicklungsumgebung vorteilhaft. Apple bietet dazu Xcode an, welchen man sich kostenlos im Mac App Store runterladen kann: <https://itunes.apple.com/de/app/xcode/id497799835?mt=12>.



Da allerdings nicht jeder ein Mac Gerät zur Verfügung hat, wird in der Live Demo mit einem Online Compiler gearbeitet: <https://swift.sandbox.bluemix.net/#/repl>.



```
1 /** Basic Fibonacci sequence in Swift.
2  * Demonstrates basic functional recursion.
3  */
4
5 func fibonacci(_ number: Int) -> Int {
6     assert(number > 0 && number < 1000, "Must choose a number between 0 and 1000.")
7
8     if number <= 2 {
9         return 1
10    } else {
11        return fibonacci(number - 1) + fibonacci(number - 2)
12    }
13 }
14
15 print(fibonacci(22))
16
17
```

Swift Ver. 3.1.1 (Release)
Platform: Linux (x86_64)

17711

New Load Code Save Code Settings

Während die Code Snippets im Einzelnen erklärt werden, kann der Nutzer somit im Compiler diese Testen und mit der Sprache vertraut werden.

Semikolon

Wie man in den folgenden Codesnippets sehen kann, wird bei Swift kein Semikolon am Ende eines Ausdrucks benötigt. Dies ist nur der Fall, wenn mehrere Ausdrücke in einer Zeile vorhanden sind, um diese voneinander abzutrennen.

Variablen und Konstanten

Zur Erstellung von Variablen hat der User zwei Möglichkeiten. Zum einem kann die Variable als var erstellt werden oder als Konstante mit let. Der Unterschied besteht darin, dass sich eine Konstante nachdem sie einmal definiert wurde nicht mehr ändern kann. Es wird dazu geraten, wenn möglich eine Variable als let zu deklarieren.

```
//Variablen  
var str = "Hallo"  
str = "Tschüss"
```

Nicht möglich, da der
Konstante „name“ schon
ein Wert zugewiesen
wurde

```
//Konstanten  
let name = "Katharina"  
name = "Andre"
```

Type Inference & Type Annotation

Da bei Swift die Variablen mit var und let deklariert werden, folgt die Bestimmung des Variablen Typs automatisch. Diese Typenbestimmung kann nach der Deklaration nicht mehr geändert werden. Falls eine Variable oder Konstante nun Deklariert werden soll, aber die Wertzuweisung erst im Nachhinein stattfinden soll, muss man dieser direkt einen Typ zuordnen.

```
//Type Inference  
var age = 19 //Int  
var pi = 3.14 //Double  
var name = "Katharina" //String  
age = true
```

Nicht möglich, da die
Variable „age“ vom Typen
Int ist, und somit keinen
bool Wert besitzen kann.

```
//Type Annotation  
var heightInCm  
var heightInCm: Int  
heightInCm = 172
```

Nicht möglich, da die
Variable „heightInCm“
keinem Typ zugeordnet
wurde

```
let surname: String  
surname = "Heer"  
surname = "Harbrecht"
```

Nicht möglich, da der
Konstante „surname“
schon ein Wert
zugewiesen wurde.

Datentypen

Wie in so gut wie allen Programmiersprachen gibt es verschiedene Datentypen, die verwendet werden können. Im folgenden Codesnippet sind die vier wichtigsten mit

Beispiel aufgelistet, sowie ein Beispiel mit if-Anweisung gezeigt um diesen einen Wert zuzuweisen.

```
//String  
var name = "Katharina"
```

```
//Int  
var age = 19
```

```
//Double  
var pi = 3.14
```

```
//Bool  
let isfemale = true
```

```
var xChromosomen: Int
```

```
if isfemale {  
  xChromosomen = 2  
}  
else {  
  xChromosomen = 1  
}
```

Typumwandlung

Da wie bereits bei der Type Inference & Type Annotation erklärt wurde, der Datentyp einer Variable bereits bei der Deklaration festgelegt wird, müssen gelegentlich Typumwandlungen vorgenommen werden, um beispielsweise mit den vorhandenen Variablen fehlerfrei rechnen zu können.

```
var pi = 3 + 0.141 //pi = 3.141
```

```
var aInt = 3
```

```
var aDouble = 0.141
```

```
var againpi = aInt + aDouble
```

```
var againpi = Double(aInt) + aDouble // againpi = 3.141
```

```
var withlost = aInt + Int(aDouble) // withlost = 3
```

```
var x = 3
```

```
var y = 10
```

```
var z = y / x // z = 3
```

```
var w = Double(x) + Double(y) // z = 3.33...
```

```
var n = 10/3 // n = 3
```

Nicht möglich, da sich die beiden verschiedenen Datentypen nicht miteinander verrechnen lassen.

Optionals

Bei Optionals handelt es sich um Variablen, bei welchen unklar ist, ob diese überhaupt gesetzt werden. Ein Beispiel dazu ist ein Errorcode. Die Variable errorcode existiert, aber abhängig davon ob ein Fehler auftritt oder nicht wird diese gesetzt oder behält den wert „null“. Wenn man versucht Optionals als diese auszulesen, werden sie als Optional(wert) angezeigt. Um dann an den Wert als solches ranzukommen gibt es verschiedene Möglichkeiten, welche im folgenden Codesnippet gezeigt werden.

```
var errorcode: Int?  
errorcode = 404  
print(errorcode) // Optional(404)
```

Optionals können mit einem Fragezeichen nach dem Datentyp definiert werden

```
//forced unwrapping  
if errorcode != nil {  
  print(errorcode!) // 404  
}
```

Optionals können mit einem Ausrufezeichen nach dem Datentyp definiert werden.

```
//implicit unwrapping  
var errormessage: String!  
errormessage = nil  
if errormessage != nil {  
  print(errormessage)  
}
```

```
// guard  
func addiere (x:Int?, y:Int?)->Int?  
{  
  guard let x = x else { return nil }  
  guard let y = y else { return nil }  
  return x+y  
}
```

```
//if let  
var age: Int?  
age = 19  
if let myage = age {  
  print(myage) //19  
}
```

Funktionen

Um eine Funktion zu deklarieren wird diese in geschweifte Klammer gepackt, sodass der Inhalt beim Aufruf abgearbeitet werden kann.

```
func addiere(x:Int,y:Int)->Int {  
  return x+y  
}
```

```
var summe = addiere(x:1,y:5) //summe = 6
```

```
func fibonacci(x:Int)->Int {  
  if x==1 { return 1 }  
  else if x==0 { return 0 }  
  else {  
    return (fibonacci(x:x-1)+fibonacci(x:x-2))  
  }  
}
```

```
var fibo22 = fibonacci(22) //fibo22 = 17711
```