

**La guida pratica per diventare  
un programmatore iOS di successo**

# Creare applicazioni per iPhone e iPad con **SWIFT**



**Aggiornata all'ultima versione**

**Roberto Travagliante**

**Copyright © 2014 - Roberto Travagliante**

**Aggiornamento e revisione: Giugno 2016**

*Email: [info@travagliante.com](mailto:info@travagliante.com)*

*Sito web: <http://www.travagliante.com>*

*Tutti i diritti sono riservati a norma di legge e a norma delle convenzioni internazionali. Nessuna parte di questo libro può essere riprodotta in nessuna forma e con qualsiasi mezzo, elettronico, meccanico o altro, senza il permesso scritto dell'autore.*

*I contenuti di questo libro sono depositati presso gli organi competenti.*

*I nomi e i marchi citati nelle pagine di questo libro sono generalmente depositati o registrati presso le rispettive case produttrici.*

*ISBN: 978-88-907270-9-2*

*Prima edizione: Settembre 2014*

*Ultimo aggiornamento e revisione: Giugno 2016*

*“Un’idea che non trova posto a sedere è capace di fare la rivoluzione.”*

*Leo Longanesi*

Questo libro è dedicato a te, che hai una grande idea e vuoi mostrarla al mondo, trasformandola in un'app.

# Indice

<b>Premessa.....</b>	<b>8</b>
<b>Informazioni su questo aggiornamento.....</b>	<b>10</b>
<b>Cosa è Swift.....</b>	<b>11</b>
<b>La sintassi di Swift.....</b>	<b>13</b>
<b>Capitolo 1 - I tipi di dati di Swift .....</b>	<b>15</b>
Le Costanti e le Variabili .....	15
Conversione di un tipo in un altro .....	16
Array .....	17
Dizionari .....	18
<b>Capitolo 2 - Il controllo del flusso .....</b>	<b>20</b>
La struttura selettiva IF .....	20
Controllo di flusso: selezione multipla SWITCH .....	21
Il ciclo WHILE .....	22
Il ciclo REPEAT...WHILE .....	23
Il ciclo FOR .....	23
La struttura FOR...IN .....	24
<b>Capitolo 3 - Le funzioni.....</b>	<b>26</b>
Dichiarare una funzione .....	26
Funzioni con numero variabile di argomenti .....	27
Funzioni che restituiscono più valori .....	27
Funzioni annidate .....	28
Ancora sulle funzioni .....	29
<b>Capitolo 4 - Classi e Oggetti .....</b>	<b>30</b>
Dichiarazione di una classe .....	30
Istanza di una classe: gli Oggetti .....	31
Costruttori e Distruttori .....	31
Metodi Getter e Setter .....	33
Ereditarietà .....	35
Alcune curiosità sull'invocazione dei metodi.....	36
<b>Capitolo 5 - Le Strutture.....</b>	<b>38</b>
Dichiarazione di una struttura.....	38
Metodi di una struttura .....	38

Accesso alle strutture .....	39
<b>Capitolo 6 - I Protocolli.....</b>	<b>40</b>
Dichiarazione di un protocollo .....	40
Utilizzo di un protocollo.....	40
<b>Capitolo 7 - Le Estensioni.....</b>	<b>42</b>
Dichiarazione e utilizzo di un'estensione .....	42
<b>Capitolo 8 - La prima semplice App .....</b>	<b>43</b>
Prerequisiti .....	43
L'applicazione Hello World.....	45
Hello World migliorata .....	49
Creazione dell'interfaccia grafica.....	51
La gestione degli outlets .....	54
Aggiungiamo le azioni.....	57
Una nuova caratteristica di XCode: Size classes.....	61
Alcuni miglioramenti .....	70
Aggiungiamo la gestione degli errori .....	73
<b>Capitolo 9 - Una app più complessa.....</b>	<b>77</b>
Un'applicazione con più videate .....	77
Creiamo il progetto .....	78
Lo strumento della Storyboard .....	79
Aggiungere una maschera alla Storyboard .....	90
Collegare due view controllers .....	92
<b>Capitolo 10 - L'applicazione To-Do List.....</b>	<b>95</b>
Creiamo un'applicazione utile .....	95
Allineamento automatico.....	98
Aggiungiamo un Table View Controller.....	101
Aggiungiamo il Navigation Controller.....	103
Implementiamo l'aggiunta di nuovi elementi.....	105
Una Navigation Bar per la view di dettaglio .....	108
Creare un View Controller personalizzato .....	111
La classe ElementoToDo .....	115
L'override dei metodi del Table View Controller .....	117
La funzionalità Check Completed .....	121
L'aggiunta di nuovi tasks alla To-Do List.....	123
<b>Capitolo 11 - Uso della Fotocamera e della Libreria Foto .....</b>	<b>126</b>

L'accesso alla Camera dell'iPhone o dell'iPad.....	126
La Storyboard e l'interfaccia utente .....	128
Colleghiamo azioni e outlets .....	131
Implementiamo le azioni: Scatta una foto .....	133
Implementiamo le azioni: accedi alla libreria foto .....	136
I protocolli e i metodi Delegate .....	137
L'applicazione Fotocamera è pronta! .....	141
Alcuni miglioramenti .....	145
<b>Conclusioni.....</b>	<b>148</b>
<b>Tutte le guide di Roberto Travagliante.....</b>	<b>150</b>

## Premessa

Gentile lettore, lasciami indovinare: sei venuto a conoscenza di Swift e della sua semplicità e, finalmente, hai deciso di sviluppare applicazioni per iOS?

Dopo aver sentito tanto parlare di iPhone e iPad, di App Store e di sviluppatori che si sono arricchiti riuscendo a creare il proprio business da zero grazie ad un'app, hai deciso che è arrivato il momento di provarci anche tu?

Oppure sei già uno sviluppatore di applicazioni per iOS e vuoi capire come Swift potrà semplificare la tua vita?

O, ancora, vuoi saperne di più sulla programmazione con Swift, per migrare le tue app o crearne di nuove in modo facile e immediato?

Bene, se le cose stanno come credo, **questa è la guida che fa per te!**

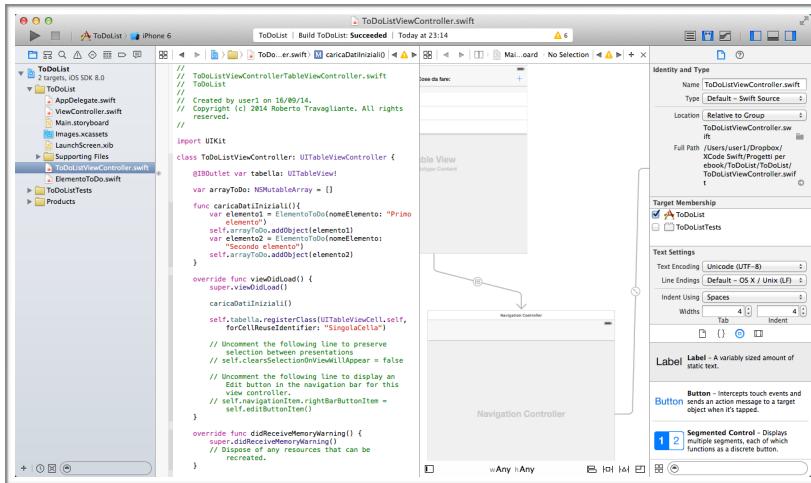
Chiariamo subito una cosa, questo non è il classico “trattato” sulla programmazione, ricco di nozioni teoriche, difficili da comprendere e da mettere in pratica per la realizzazione della tua applicazione per iPhone e iPad.

Al contrario, **questa è una guida pratica e veloce, offerta ad un prezzo estremamente vantaggioso, che si propone, in un centinaio di pagine ricche di contenuti, di mettere nelle tue mani tutti gli strumenti utili per poter essere subito “operativo” e per realizzare le tue applicazioni per iOS**, partendo dall’illustrazione della sintassi utilizzata da Swift per svolgere le più comuni operazioni, fino ad arrivare alla realizzazione di alcune applicazioni più o meno semplici, ma allo stesso tempo adeguate a farti familiarizzare con l’ambiente di sviluppo di Apple e ad aiutarti a comprendere come districarti nello sviluppo per iPhone e iPad con questo nuovo linguaggio di programmazione chiamato Swift.

Durante la lettura di queste pagine, troverai anche tante immagini che ti aiuteranno nell’apprendimento, fornendoti un supporto in più.

Inoltre, troverai tanto codice Swift (ovviamente), scritto prestando sempre massima attenzione alla chiarezza espositiva.

## Creare applicazioni per iPhone e iPad con Swift



The screenshot shows the Xcode interface with the file `ToDoListViewController.swift` open. The code implements a `UITableView` controller to manage a list of tasks. It includes methods for initializing the table view, loading data from a database, and handling user interactions like editing and deleting items.

```
// To Do List View Controller
// Roberto Travagliante - www.travagliante.com
// Created by user1 on 16/09/14.
// Copyright (c) 2014 Roberto Travagliante. All rights reserved.

import UIKit

class ToDoListViewController: UITableViewController {
    @IBOutlet var tableView: UITableView!
    var arrayToDo: NSMutableArray = []
    func caricaDBIniziali() {
        var elemento1 = ElementoToDo(nomeElemento: "Primo elemento")
        self.arrayToDo.addObject(elemento1)
        var elemento2 = ElementoToDo(nomeElemento: "Secondo elemento")
        self.arrayToDo.addObject(elemento2)
    }
    override func viewDidLoad() {
        super.viewDidLoad()
        caricaDBIniziali()
        self.tableView.registerClass(UITableViewCell.self,
            forCellReuseIdentifier: "SingleCell")
        // Uncomment the following line to preserve selection between applications
        // self.clearsSelectionOnViewWillAppear = false
        // Uncomment the following line to display an Edit button in the navigation bar for this view controller.
        // self.navigationItem.rightBarButtonItem = self.editButtonItem()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

In ogni caso, per qualsiasi eventuale chiarimento e/o approfondimento ulteriore, rispetto a quanto riportato nelle pagine di questo libro, ti invito a visitare il mio blog personale, nonché sito web di riferimento per questa e per tutte le altre mie guide <http://www.travagliante.com>.

Tramite la pagina di contatto presente sul sito, potrai contattarmi per qualsiasi necessità.

Comunque, nel ringraziarti fin da adesso per aver scelto questa guida, non ti trattengo oltre: partiamo subito “con i contenuti”!



## Informazioni su questo aggiornamento

Perché realizzare un aggiornamento di questa guida?

Il fatto è che dopo quasi due anni dal quel secondo semestre 2014 in cui Apple ha introdotto la prima versione del nuovo linguaggio Swift, sono cambiate tante cose.

Infatti, Apple ha apportato diverse modifiche e aggiornamenti che hanno interessato sia la sintassi di Swift che l'ambiente di sviluppo di Xcode.

Basti dire che nel 2015 Apple ha presentato una versione 2 del linguaggio, completamente rinnovata sia dal punto di vista della sintassi, che sotto il profilo delle funzionalità offerte.

Inoltre, nel corso del tempo, sono stati modificati alcuni aspetti dell'ambiente di sviluppo Xcode, per cui è possibile utilizzare alcune funzionalità in modo differente tra le diverse versioni.

Ad esempio, è stato notevolmente migliorato il “code completion”, per favorire lo sviluppatore durante la scrittura del codice, sono stati aggiunti nuovi strumenti per il debugging delle app, è stato inserito il supporto per nuove funzionalità (es.: il 3D Touch) e per nuovi dispositivi (es.: supporto watchOS per Apple Watch, e tvOS per la nuova Apple TV), ecc..

Al momento della stesura di questo aggiornamento, la versione corrente di Xcode è la 7.3.1 e include il supporto per la versione del linguaggio Swift 2.2 e per le seguenti versioni di sistemi operativi:

- iOS 9.3 - per lo sviluppo di applicazioni per iPhone, iPad e iPad Pro;
- watchOS 2.2 - per la creazione di app per Apple Watch;
- OS X 10.11.4 - per il sistema Mac OS. Infatti, Xcode permette di creare non soltanto applicazioni per i suoi dispositivi mobili, ma anche software in grado di funzionare con iMac, Macbook Air, Macbook Pro, ecc. (Al momento, l'ultima versione di Mac OS supportata è quella di El Capitan);
- tvOS 9.2 - per la creazione di app per la nuova Apple TV.

## Cosa è Swift

Il 10 gennaio 2007 Steve Jobs presentava al mondo la prima versione della sua “creatura”: l'**iPhone**. Un nuovo dispositivo che, insieme al proprio sistema operativo **iOS**, avrebbe di lì a poco rivoluzionato in maniera evidente il mondo del mobile computing.

Da quel giorno sono accadute tante cose. E' nato l'App Store, insieme ad un nuovo modo di concepire la realizzazione di software per telefoni cellulari. Poi è stato il momento dell'iPad, che ha dato un fortissimo impulso allo sviluppo del mercato dei tablet di nuova generazione, andando ad intaccare quella quota fino ad allora dominata dai computer e dai notebook. Infine, iOS è cresciuto, diventando il sistema operativo evoluto che conosciamo oggi, profondamente innovato anche dal punto di vista grafico.

L'ultima novità riguarda l'introduzione di questo nuovo linguaggio di programmazione, **Swift**, presentato da Apple in occasione del WWDC del 2 giugno 2014 e definito, simpaticamente, “Objective-C senza la C”.

Ciò, proprio per far comprendere l'elemento di forza del linguaggio, che semplifica enormemente la vita a tutti quegli sviluppatori che non hanno familiarità e/o dimestichezza con gli strumenti messi a disposizione dal linguaggio Objective-C.

Diciamolo chiaramente, Objective-C non è il massimo della semplicità. E' estremamente potente, ma per chi vi si avvicina per la prima volta non è facile comprenderne tutti gli aspetti in modo rapido. Ciò, anche per via della sua struttura, molto legata ad esempio all'uso di puntatori e strutture di puntatori, il “cruccio” di molti programmati, in modo particolare di quelli autodidatti, che non hanno avuto la fortuna di seguire un buon corso sul C.

In tal senso, la concorrenza è molto agguerrita. Ad esempio, il linguaggio Java utilizzato nello sviluppo per dispositivi Android è molto più semplice dal punto di vista sintattico e semantico, rispetto a Objective-C. Ancora, lo sviluppo per Windows Phone, dando la possibilità di utilizzare più linguaggi (sintatticamente più semplici) come Visual Basic o C Sharp (C#), risulta ancora più immediato.

In quest'ottica, probabilmente, Apple ha ben compreso la necessità di semplificare lo sviluppo di applicazioni per iOS per i propri developers, dando il via alla creazione di un linguaggio in grado di:

- ridurre uno dei problemi maggiori riguardante la presentazione di applicazioni all'App Store, nonché maggior causa di rigetto delle app stesse: la presenza di bugs di funzionamento intrinsechi all'errata programmazione in Objective-C (ad esempio, dovute ad un cattivo utilizzo della referenziazione e dereferenziazione di oggetti, ecc.);
- agevolare i developers, in modo da promuovere l'introduzione di nuove applicazioni in App Store, considerato che quest'ultimo frutta ad Apple e agli sviluppatori coinvolti ingenti quantità di denaro, ogni anno.

Ecco i principali motivi per cui Apple ha deciso di realizzare un nuovo linguaggio di programmazione, che si propone di rivoluzionare il mondo dello sviluppo iOS, avvicinando un numero sempre maggiore di programmatore al mondo degli "iDevices".

Come già detto, questo libro non è un trattato sul linguaggio Swift, ma una guida pronta all'uso. Divideremo il lavoro in due parti principali.

Nelle prossime pagine, faremo un excursus delle principali caratteristiche del linguaggio Swift e ne analizzeremo le maggiori potenzialità.

Nella seconda parte, invece, creeremo praticamente una semplice applicazione con Xcode e con Swift, poco più complessa della classica applicazione "Hello World!", ma non troppo! :)

In pratica, **al termine di questa guida, avrai tutte le nozioni che ti serviranno per "camminare con le tue gambe" e realizzare la tua idea, trasformandola in un'app!**

.....  
.....  
.....  
.....  
.....  
.....

**NOTA BENE:**

Questo documento è un estratto del libro

## **Creare applicazioni per iPhone e iPad con Swift**

La versione completa di quest'opera è disponibile:

Sul sito web dell'autore: <http://www.travagliante.com>

(pagina: <http://www.travagliante.com/creare-applicazioni-per-iphone-e-ipad-con-xcode-e-swift-la-guida/>)

e in tutte le più importanti librerie digitali e market on-line

.....  
.....

---

## La sintassi di Swift

In questa prima parte, analizzeremo le principali forme sintattiche del linguaggio Swift.

Per tutti coloro che conoscono già Objective-C, sarà facile rendersi conto di come Swift metta a disposizione dello sviluppatore una sintassi estremamente semplificata.

Questa semplicità è evidente fin da subito. Ad esempio, mettendo a confronto Objective-C e Swift, riguardo alla semplice assegnazione di una variabile di tipo stringa, notiamo che con Objective-C siamo costretti ad utilizzare l'operatore di dereferenziazione “`**`” e l'operatore “`@`”, in questo modo:

```
NSSString *stringa1 = @“Hello, World!”;
```

Invece, con Swift, la sintassi appare molto più snella:

```
var stringa1 = “Hello, World!”
```

L'aspetto importante è che tale semplicità è introdotta senza limitare il programmatore, durante la sua attività di sviluppo. Anzi, sotto alcuni punti di vista, bisogna rilevare che Swift introduce evidenti miglioramenti, che ne mettono in luce le potenzialità.

Ad esempio, in merito ai tipi di dati, Swift fornisce tutti i tipi di dati fondamentali, previsti dall'Objective-C, come ad esempio:

- “`Int`” per gli interi;
- “`Float`” o “`Double`” per i numeri in virgola mobile;
- “`String`” per le stringhe (le sequenze alfanumeriche);
- “`Bool`” per i valori booleani.

Inoltre, Swift mette a disposizione versioni più flessibili e potenti per la gestione degli insiemi di dati come quelli gestiti dai tipi “`Array`” e “`Dictionary`”.

Ancora, per la visualizzazione a video delle variabili, è possibile ricorrere a due forme. Supponiamo di voler visualizzare una variabile pippo. Possiamo utilizzare una chiamata alla funzione “`print()`”, in questo modo:

```
print("\u0028pippo\u0029")
```

oppure, semplicemente, digitare:

```
pippo
```

in modo molto più semplice e immediato.

Niente male, vero?

Detto questo, vediamo subito le principali caratteristiche del linguaggio Swift.

.....  
.....  
.....  
.....  
.....  
.....  
.....

NOTA BENE:

Questo documento è un estratto del libro

## **Creare applicazioni per iPhone e iPad con Swift**

La versione completa di quest'opera è disponibile:

Sul sito web dell'autore: <http://www.travagliante.com>

(pagina: <http://www.travagliante.com/creare-applicazioni-per-iphone-e-ipad-con-xcode-e-swift-la-guida/>)

e in tutte le più importanti librerie digitali e market on-line

.....  
.....

---

# Capitolo 1 - I tipi di dati di Swift

## Le Costanti e le Variabili

Come ogni linguaggio di programmazione, anche Swift prevede una precisa sintassi per la dichiarazione di costanti e di variabili.

Ricordiamo che per **costante**, si intende un valore che una volta dichiarato non può essere modificato. Invece, per **variabile** si intende un nome simbolico al quale assegnare un valore iniziale, che può essere successivamente modificato (es.: incrementato, decrementato, ecc.).

Sia le costanti che le variabili sono utilizzate nel corso dell'elaborazione, da parte della nostra app.

Come si dichiara una costante in Swift? Si dichiara con una istruzione “*let*”:

```
let costante1 = 23
```

L'istruzione indicata, ad esempio, assegna il valore 23 alla costante chiamata “*costante1*”. Come già detto, dopo la dichiarazione di una costante, non è possibile assegnare un nuovo valore a “*costante1*”.

Per le variabili, invece, si utilizza l'istruzione “*var*”:

```
var variabile1 = 23  
variabile1 = 50  
variabile1 = variabile1 + 5
```

Nota bene che in questo esempio non abbiamo indicato il tipo di “*variabile1*”.

In effetti, Swift è un linguaggio che, a differenza di Objective-C, non impone di dichiarare esplicitamente il tipo delle variabili.

In particolare, il tipo di una variabile viene determinato in base al valore assegnato tramite l'istruzione “*var*”. Nell'esempio riportato

sopra, la prima istruzione definisce una variabile “variabile1” di tipo “Int” (intero) e vi assegna il valore 23. Tutte le successive operazioni su tale variabile avvengono utilizzando “variabile1” come un intero.

Comunque, con Swift, è possibile anche dichiarare esplicitamente il tipo di una variabile, con la sintassi:

“var <nomevariabile>: <tipo>”.

Ad esempio:

```
var variabile1: Int  
var variabile2: Double  
var variabile3: String
```

La dichiarazione esplicita del tipo di una variabile è utile:

- ogni qualvolta il valore iniziale assegnato non fornisce abbastanza informazioni sul proprio tipo;
- ogni volta in cui non viene assegnato alcun valore iniziale.

Tanto per richiamare un caso pratico, se vogliamo dichiarare una variabile in grado di contenere un numero decimale in virgola mobile, al quale assegnare il valore 50 (un valore intero), possiamo scrivere il seguente codice, utilizzando la dichiarazione esplicita del tipo:

```
var variabile1: Float = 50
```

oppure, possiamo dichiarare implicitamente il tipo float, nel seguente modo:

```
var variabile1 = 50.0
```

## Conversione di un tipo in un altro

Una volta definito il tipo di una variabile, questo non viene più modificato in modo隐式. Ciò, tutela lo sviluppatore dal commettere eventuali errori in fase di programmazione.

Tuttavia, può essere necessario effettuare una conversione dei dati in un tipo differente, in base al tipo di operazione e/o di elaborazione che si intende compiere.

Ad esempio, supponiamo di voler assegnare ad una costante “*anni*” la nostra età e di voler successivamente assegnare ad una nuova costante “messaggio”, la stringa “Complimenti, hai <N> anni!”.

Possiamo utilizzare le seguenti istruzioni:

```
let msg1 = "Complimenti, hai "
let msg2 = " anni!"
let anni = 23
```

e, infine, comporre il messaggio, convertendo in modo esplicito la variabile “*anni*” in un tipo Stringa contenente il valore “23”:

```
let messaggio = msg1 + String(anni) + msg2
```

Se non utilizzassimo l’operatore di casting String(*anni*), otterremmo un errore in fase di compilazione.

Lo stesso risultato possiamo ottenerlo scrivendo soltanto 2 righe e inserendo il valore da convertire tra parentesi tonde precedute da un backslash “\”:

```
let messaggio = "Complimenti, hai \(anni) anni!"
```

## Array

Come ogni linguaggio di programmazione ad alto livello, Swift prevede la possibilità di gestire strutture dati più complesse, come Array e Dizionari.

Per creare un Array, si utilizzano le parentesi graffe “[ ]”.

Per esempio, per creare un array contenente i nomi delle 4 stagioni:

```
var stagioni = ["primavera", "estate", "autunno", "inverno"]
```

Invece, per creare un array vuoto, vale a dire senza elementi, di tipo stringa:

.....  
.....  
.....  
.....  
.....

**NOTA BENE:**

Questo documento è un estratto del libro

## **Creare applicazioni per iPhone e iPad con Swift**

La versione completa di quest'opera è disponibile:

Sul sito web dell'autore: <http://www.travagliante.com>

(pagina: <http://www.travagliante.com/creare-applicazioni-per-iphone-e-ipad-con-xcode-e-swift-la-guida/>)

e in tutte le più importanti librerie digitali e market on-line

.....  
.....

---

## Il ciclo REPEAT...WHILE

Quando è necessario fare in modo che il ciclo sia eseguito almeno una volta, bisogna ricorrere alla struttura REPEAT...WHILE, che letteralmente dice “fai... finché la condizione non è vera”. Eccone un esempio:

```
var num = 0
repeat {
    num = num + 1
    print("\(num)")
}
while num < 10
```

Questo codice stamperà a video i numeri da 1 a 10.

Nota bene: se la variabile “num” fosse valorizzata già con 10, anziché con 0, il codice illustrato eseguirebbe comunque almeno una volta il ciclo, visualizzando a video il numero 11 (perché prima della stampa a video la variabile “num” viene incrementata di 1).

## Il ciclo FOR

Quando il numero di iterazioni è predeterminato, vale a dire quando si conosce a priori per quante volte deve essere eseguito un determinato codice, è possibile utilizzare una struttura di tipo FOR.

La dichiarazione di un ciclo for, con la notazione classica, prevede l’indicazione di un valore iniziale, di una condizione, e dello step (o passo, o incremento). Ad esempio:

```
for var i = 0; i < 10; i++ {
    print("\(i)")
}
```

mostra a video i numeri da 0 a 9.

Tuttavia, con l’introduzione di Swift 2, l’operatore di incremento classico “`++`” utilizzato in un ciclo FOR è deprecato e sarà

abbandonato definitivamente a partire da Swift 3. Per questo motivo, conviene abituarsi subito alla cosiddetta “notazione abbreviata”.

Con la notazione abbreviata, otteniamo la stessa cosa nel modo seguente:

```
for i in 0...9 {  
    print("\(i)")  
}
```

oppure così:

```
for i in 0..<10 {  
    print("\(i)")  
}
```

Se vogliamo visualizzare soltanto i numeri pari, dobbiamo fare aumentare la variabile indice di 2 alla volta:

```
for var i = 0; i < 10; i=i+2 {  
    print("\(i)")  
}
```

Tuttavia, dobbiamo tenere sempre conto che questa notazione in stile linguaggio C è deprecata e sarà presto abbandonata, con l'introduzione di Swift 3.

## La struttura FOR...IN

Quando il numero di iterazioni è si predeterminato, ma in relazione ad un insieme di dati (es.: il numero di elementi di un array), possiamo ricorrere alla struttura FOR...IN.

Ad esempio, se abbiamo un array di interi e vogliamo visualizzare soltanto i numeri maggiori di 10, allora possiamo scrivere il seguente codice:

```
let arrayNumeri = [3, 12, 5, 8, 16]  
for valore in arrayNumeri {
```

```
if valore > 10 {  
    print("\(valore)")  
}  
}
```

Possiamo utilizzare la struttura FOR...IN anche con un insieme di dati di tipo Dictionary, utilizzando due for annidati, vale a dire iterando prima all'interno delle chiavi e poi all'interno dei valori per ciascuna chiave.

.....  
.....  
.....  
.....  
.....  
.....  
.....

NOTA BENE:

Questo documento è un estratto del libro

## **Creare applicazioni per iPhone e iPad con Swift**

La versione completa di quest'opera è disponibile:

Sul sito web dell'autore: <http://www.travagliante.com>

(pagina: <http://www.travagliante.com/creare-applicazioni-per-iphone-e-ipad-con-xcode-e-swift-la-guida/>)

e in tutte le più importanti librerie digitali e market on-line

.....  
.....

---

# Capitolo 8 - La prima semplice App

## Prerequisiti

Dopo aver illustrato gli elementi principali della sintassi Swift, passiamo allo sviluppo vero e proprio di una semplice, ma rappresentativa applicazione, in grado di funzionare su un dispositivo iOS (iPhone o iPad).

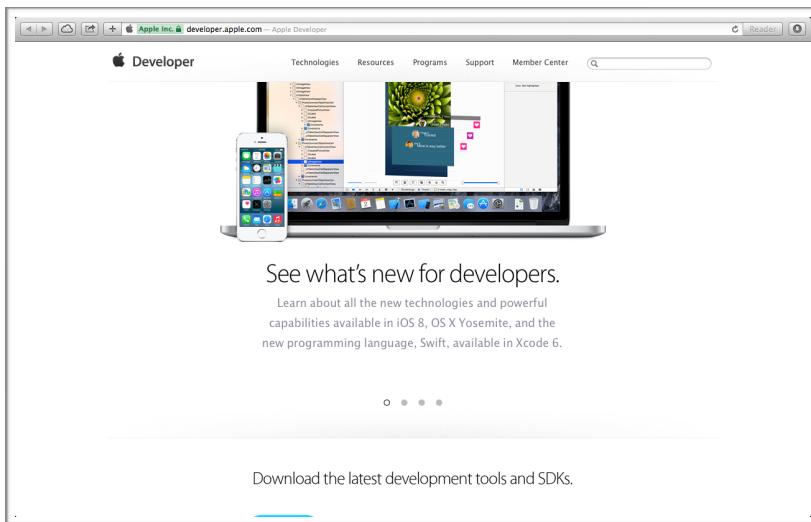
Prima di andare avanti, però, abbiamo bisogno, se non lo abbiamo già fatto, di installare l'ambiente di sviluppo Xcode.

Xcode è l'ambiente di sviluppo integrato (IDE) fornito da Apple a tutti coloro che vogliono sviluppare applicazioni in grado di “girare” sui propri sistemi operativi.



Esso non consente soltanto di sviluppare applicazioni per iPhone e iPad, ma permette anche di realizzare programmi, librerie e componenti di vario genere per il sistema operativo Mac OS X.

E' importante evidenziare che Apple fornisce questo IDE in modo gratuito, tramite accesso al portale per gli sviluppatori Apple Developer <https://developer.apple.com> (dove è possibile reperire anche le versioni beta, non ancora rilasciate al pubblico, comprendenti le librerie più recenti), oppure mediante accesso al Mac App Store, effettuato con qualsiasi sistema (iMac o MacBook) equipaggiato con Mac OS.



Questo è il link diretto, per il Mac App Store italiano: <https://itunes.apple.com/it/app/xcode/id497799835?mt=12>.

Al momento della stesura di questa guida, la versione di Xcode disponibile è la 6, appena rilasciata in versione definitiva.

Questa versione è la prima che supporta l'impiego del linguaggio Swift.



Dopo avere installato l'applicazione Xcode sul nostro Mac, possiamo procedere con lo sviluppo della nostra prima app.

### L'applicazione Hello World

Come avviene per il 99% dei manuali pratici sulla programmazione, svilupperemo un'app in grado di visualizzare la scritta “Hello, world!”

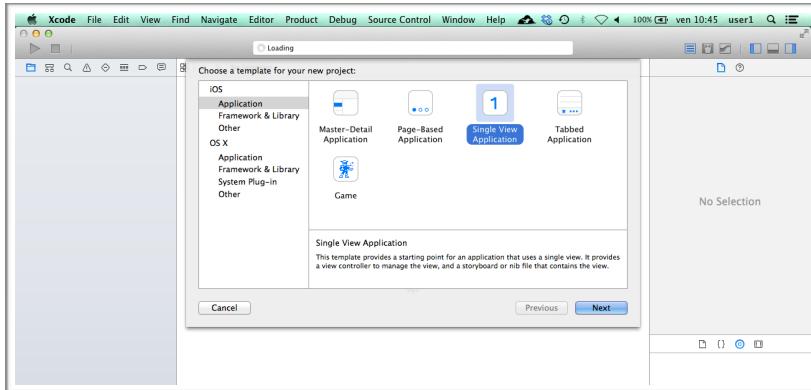
E’ ovvio che un’applicazione di questo tipo è assolutamente inutile dal punto di vista dell’utilizzo pratico. Tuttavia, essa permette di familiarizzare con l’ambiente di sviluppo e con le sue funzionalità e, allo stesso tempo, di comprenderne gli aspetti più elementari.

Inoltre, il funzionamento di questa app ci permette di verificare anche se abbiamo configurato correttamente l’ambiente di sviluppo.

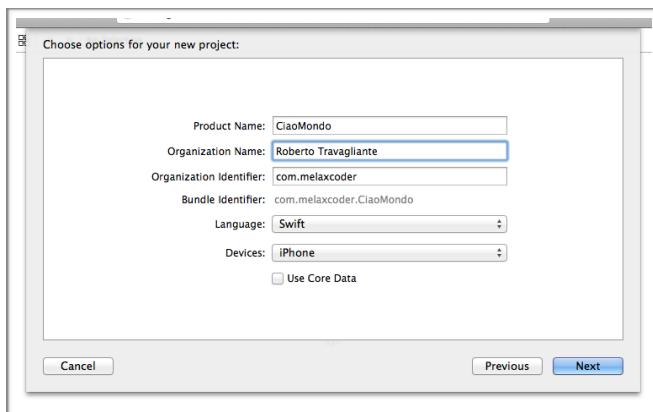
La nostra applicazione farà una sola cosa: visualizzare la stringa “Hello, world!” nella console di XCode.

Detto questo, avviamo l’applicazione Xcode installata e creiamo un nuovo progetto (selezionare il menu File -> New -> New

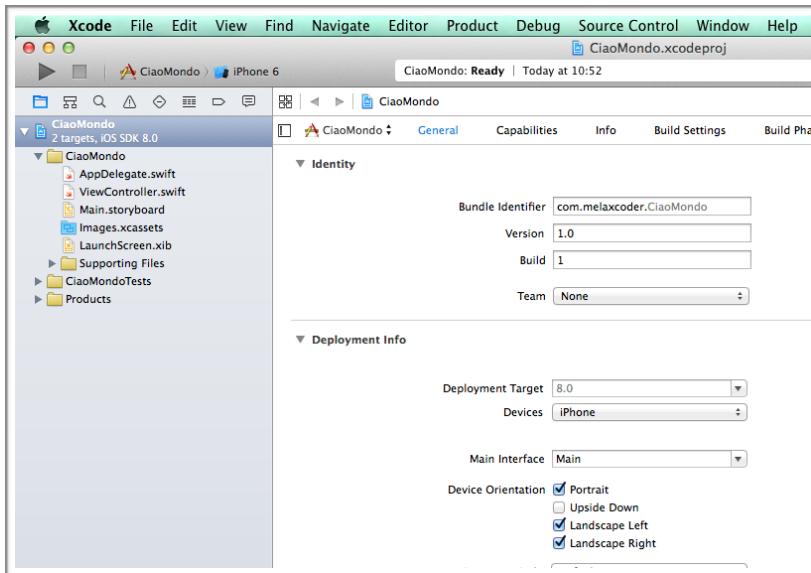
Project, oppure, da tastiera, usare la combinazione ⌘⌘N), scegliendo il template più semplice tra quelli offerti per le applicazioni iOS: “Single View Application” e cliccando sul pulsante “Next”:



A questo punto arriva la parte più importante: nella schermata che ci appare, scegliamo di utilizzare il linguaggio Swift. Pertanto, oltre alle altre opzioni (es.: nome dell'applicazione, organizzazione, ecc.), accertiamoci di selezionare “Swift” in corrispondenza della voce “Language”.



Cliccando ancora sul pulsante “Next”, verrà generato un progetto riguardante un’applicazione che non fa nulla, ma perfettamente funzionante.



Tra i files presenti nel riquadro inherente la gerarchia del progetto, troviamo un file denominato “AppDelegate.swift”.

Aprendo questo file, troveremo il codice riguardante la definizione di una classe “AppDelegate”, con una serie di dichiarazioni di funzioni predisposte dall’IDE per la modifica da parte nostra (per maggiori dettagli sul loro utilizzo, leggere i rispettivi commenti).

In corrispondenza della funzione “application()”, troviamo un commento che ci indica il punto a partire dal quale personalizzare la nostra app:

// Override point for customization after application launch.

Subito sotto questa riga, inseriamo il codice per visualizzare del testo in console (funzione print() )

```
print("Hello, world!")
```

Prima dell'introduzione di Swift 2, erano previsti due comandi, per la visualizzazione del testo in console:

- **println**, in grado di visualizzare il testo seguito da un ritorno a capo (newline);
- **print**, in grado di visualizzare il testo senza il carattere di newline.

A partire da Swift 2, il comando **println** è stato abolito. Attualmente, **print** svolge le funzioni del precedente println, quindi mostra il ritorno a capo. Per non visualizzare il carattere di newline, è possibile utilizzare il comando seguente:

```
print("Hello, world!", terminator: "")
```

Il file AppDelegate.swift, al termine della nostra modifica, dovrebbe apparire grosso modo come illustrato nella seguente screenshot:

The screenshot shows the Xcode code editor with the AppDelegate.swift file open. The code contains the following content:

```
// CiaoMondo
// Created by user1 on 12/09/14.
// Copyright (c) 2014 Roberto Travagliante. All rights reserved.

import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions: [NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.

        println("Hello, world!")

        return true
    }

    func applicationWillResignActive(application: UIApplication) {
        // Sent when the application is about to move from active to inactive state. This can occur for
        // certain types of temporary interruptions (such as an incoming phone call or SMS message) or when
        // the user quits the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates.
        // Games should use this method to pause the game.
    }

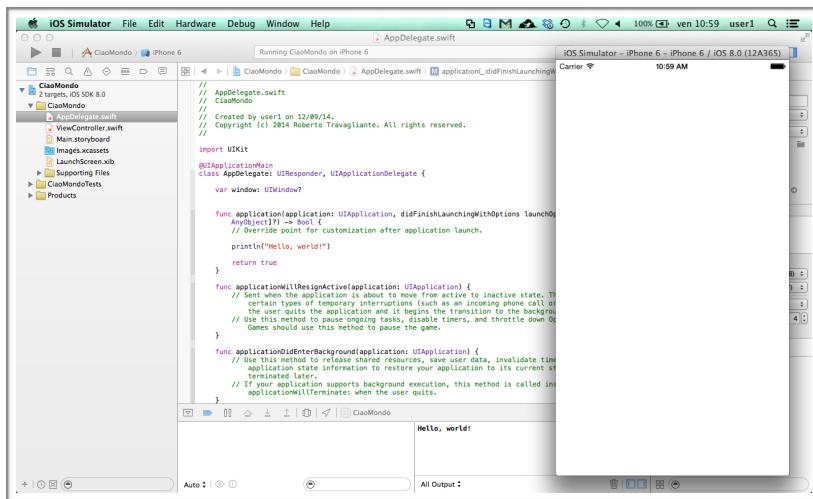
    func applicationWillEnterBackground(application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers, and store enough
        // application state information to restore your application to its current state in case it is
        // terminated later.
        // To handle application delegate background execution, this method is called instead of.
    }
}
```

The right side of the Xcode interface shows the Assistant Editor with the storyboard file "CiaoMondo.storyboard". A sidebar panel on the right is titled "Text Setting:" and includes options for "Text Encoding", "Line Endin", "Indent Usi", and "Widt".

OK, adesso possiamo verificare il funzionamento della nostra semplice applicazione.

Facendo click sul pulsante “Run” (pulsante con il simbolo triangolare Play in alto a sinistra), dovremmo vedere avviarsi un’app vuota nell’emulatore iPhone e la scritta “Hello, world!” dovrebbe apparire sul riquadro di Xcode riguardante la console (situato in XCode, generalmente in basso, salvo personalizzazioni dell’IDE).

E’ importante notare che non viene mostrato nulla nell’emulatore, in quanto abbiamo scritto codice per visualizzare qualcosa non su quest’ultimo, bensì sullo standard output (nel nostro caso rappresentato dalla console di XCode).



Nella prossima app vedremo come visualizzare questa scritta sull’emulatore, realizzando un’app leggermente più complessa di quella vista e utilizzando un componente chiamato “label”.

## Hello World migliorata

Adesso passiamo alla creazione di una nuova app, che visualizza una scritta qualsiasi (o anche semplicemente “Hello, world!”) direttamente sull’emulatore iOS.

.....  
.....  
.....  
.....  
.....  
.....  
.....

NOTA BENE:

Questo documento è un estratto del libro

## **Creare applicazioni per iPhone e iPad con Swift**

La versione completa di quest'opera è disponibile:

Sul sito web dell'autore: <http://www.travagliante.com>

(pagina: <http://www.travagliante.com/creare-applicazioni-per-iphone-e-ipad-con-xcode-e-swift-la-guida/>)

e in tutte le più importanti librerie digitali e market on-line

.....  
.....

---

Partiamo dal template “Single View Application”, clicchiamo sul pulsante “Next”, diamo un nome alla nostra app e, soprattutto, selezioniamo il linguaggio “Swift”. Clicchiamo ancora una volta sul pulsante “Next”, scegliamo la directory in cui Xcode dovrà creare i files del progetto e salviamo il tutto, in modo da trovarci con il nostro progetto vuoto, pronto per lo sviluppo.

A questo punto, apriamo il file “Main.storyboard”. Prima della realizzazione dello strumento della Storyboard da parte di Apple, per mostrare, da una view, un’altra view, era necessario svolgere tutta una serie di attività (collegamento di outlets, chiamata da codice di metodi del view controller, definizione di animazioni, ecc.) che mettevano in forte difficoltà gli sviluppatori alle prime armi e che adesso sono state enormemente semplificate.

Personalmente, quando ho iniziato a studiare Xcode e l’Objective-C, ho avuto una certa difficoltà a comprendere il rapporto tra le views e i view controllers e, in particolare, quella di fare apparire una seconda view, cliccando un pulsante presente sulla prima view, la ricordo come una grande “conquista”!

Adesso, ogni volta che vedo la Storyboard di Xcode penso: “Ma non potevano inventarla prima?”.

## **Lo strumento della Storyboard**

Va evidenziato che la Storyboard è disponibile indipendentemente dal linguaggio selezionato e, pertanto, la troviamo anche se scegliamo di sviluppare un’app in Objective-C.

La Storyboard incorpora in sé un altro strumento di Xcode: l’Interface Builder. Si tratta di uno degli strumenti più importanti di Xcode, che permette di creare le interfacce utente della nostra app, utilizzando il drag & drop per disporre, ad esempio, i diversi outlets (pulsanti, label, text fields, sliders, ecc.) nella view e definendo tutte le proprietà dei diversi componenti della nostra UI. L’impiego simultaneo di Interface Builder e della Storyboard permette allo sviluppatore un controllo molto efficace sull’interfaccia dell’app che intende realizzare.

In effetti, con la Storyboard, creare un'app come quella che vogliamo realizzare noi, diventa molto facile.

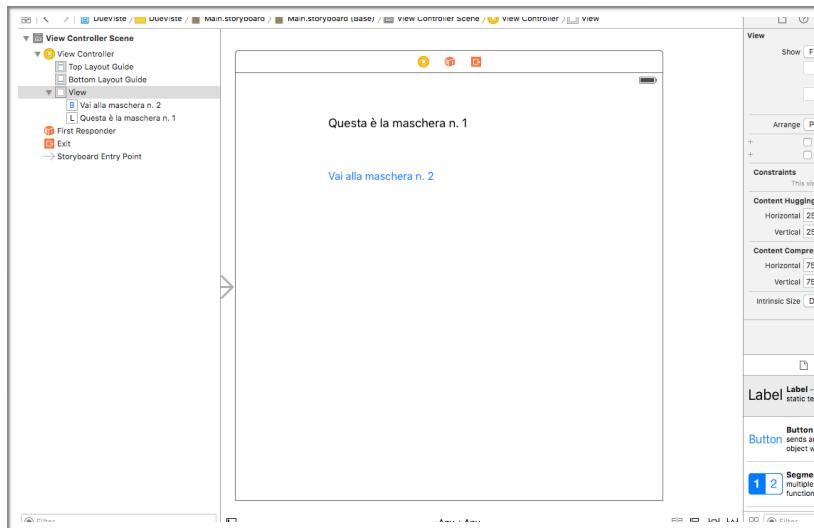
Vediamo subito come fare, dal punto di vista pratico. Qui non ci soffermiamo sulla qualità grafica della nostra applicazione, ma cercheremo comunque di rivedere la gestione dei constraints, già vista nel capitolo precedente, perché ritengo che sia molto importante e utile.

Stavolta, nella visualizzazione, attiviamo la modalità “wAny hAny” (per maggiori dettagli, vedi il capitolo precedente: Size classes). Ciò, ci permetterà di creare un'app che supporti tutte le tipologie dei dispositivi (iPhone e iPad di qualsiasi dimensione e orientamento).

Come già detto, apriamo il file “Main.storyboard” e aggiungiamo, nel view controller:

- una label con il testo “*Questa è la maschera n. 1*”;
- un pulsante con il testo “*Vai alla maschera n. 2*”.

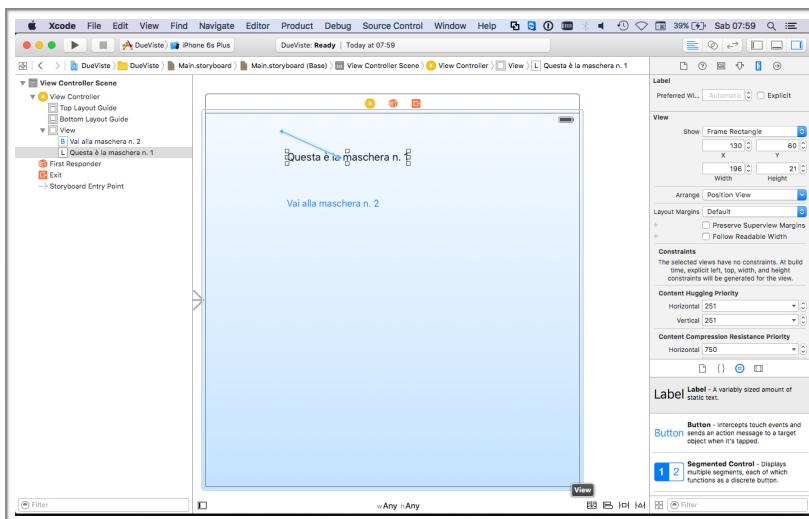
L'aspetto del view controller dovrebbe apparire più o meno come segue:



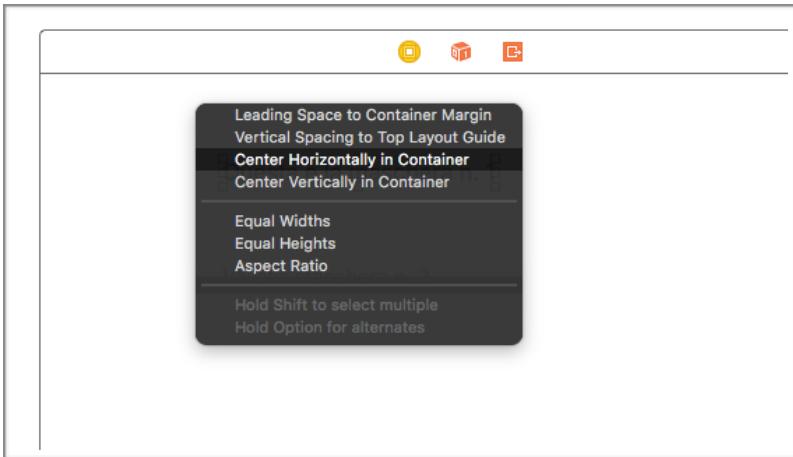
Aggiungiamo i constraints, in modo da ottenere un'adeguata visualizzazione per qualsiasi dispositivo. In particolare, facciamo in modo da disporre sia la label che il pulsante al centro della view, ad una distanza di:

- 50 punti dal margine superiore, per la label;
- 40 punti dalla label, per il pulsante.

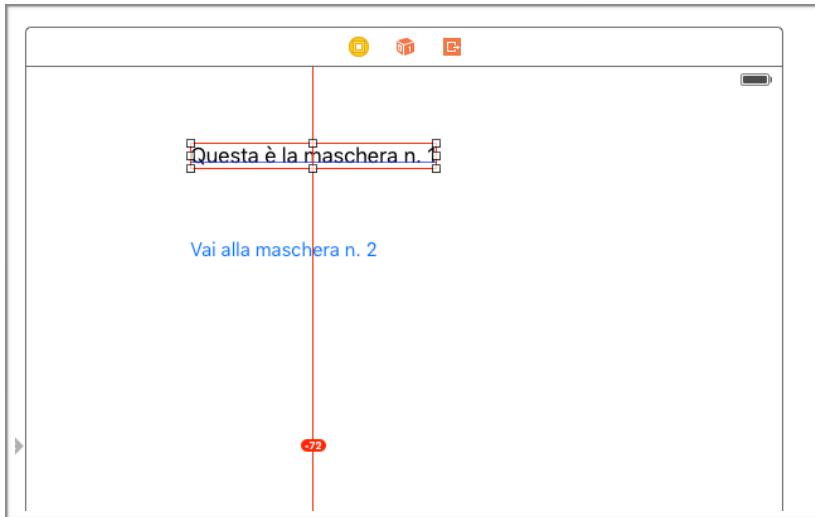
Iniziamo con il disporre sia label che pulsante al centro della view. Per farlo, selezioniamo prima la label e poi, tenendo premuto il tasto CTRL, clicchiamo all'interno della label stessa e trasciniamo il puntatore del mouse fino alla view (all'esterno della lable, ma all'interno della finestra):



Apparirà un popup, che ci permetterà di allineare la label al suo container, vale a dire la view:



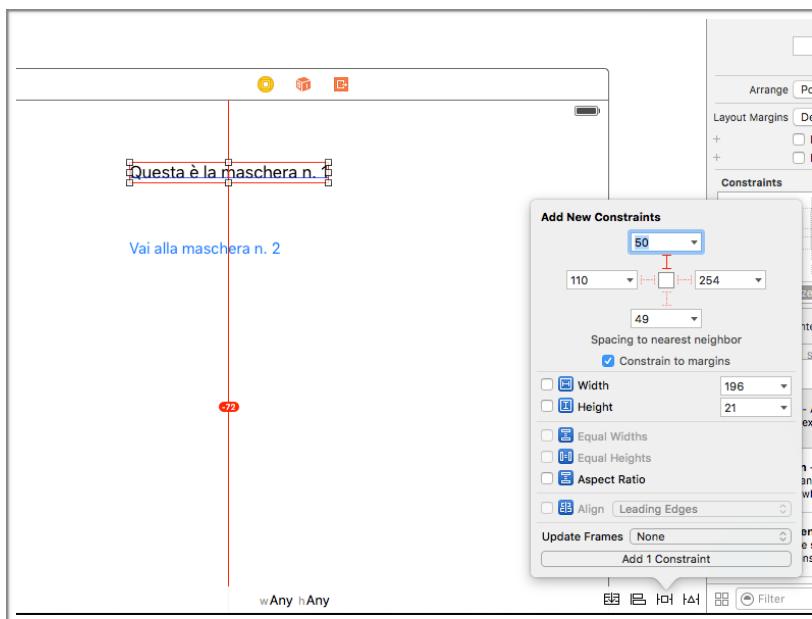
Selezioniamo la voce “Center Horizontally in Container”. A questo punto, Xcode ci mostrerà un errore, informandoci tramite una linea rossa che c’è un constraint mancante. Più precisamente, abbiamo stabilito il constraint orizzontale, ma non abbiamo detto a Xcode come disporre la label verticalmente:



Inoltre, il constraint inserito, non coincide con la posizione corrente della label nella view, discostandosi di -72 punti dalla posizione centrale. Prima di pensare a quest'altro problema, pensiamo al constraint verticale.

Abbiamo detto che vogliamo stabilire che la label sia distante 50 punti dal margine superiore. Quindi, aggiungiamo un altro constraint.

Clicchiamo sull'apposito pulsante presente in basso a destra, come abbiamo visto nel capitolo precedente e, nell'apposito popup, inseriamo il valore 50, così come illustrato nella seguente screenshot, clicchiamo nella piccola linea rossa relativa alla distanza superiore e aggiungiamo il constraint:



Fatto ciò, la situazione cambierà. Le linee rosse, indice di errore, spariranno e saranno sostituite da linee gialle, che indicano

dei warnings, vale a dire errori risolvibili tramite gli strumenti automatici di Xcode.

Più precisamente, selezionando l'outlet al quale abbiamo aggiunto i constraints (la label) noteremo:

- delle linee gialle continue, che rappresentano i constraints che abbiamo inserito, che mostrano lo scostamento in punti rispetto alla posizione attuale dell'outlet (in figura: -10 e -72);
- delle linee gialle tratteggiate, che rappresentano la posizione in cui Xcode si aspetterebbe di trovare l'outlet, in base ai constraints inseriti.



Possiamo risolvere questi problemi direttamente tramite Xcode, oppure manualmente, come abbiamo già visto. Per risolvere velocemente, usiamo gli strumenti di Xcode.

Clicchiamo, come già fatto in precedenza, sul piccolo pulsante giallo posto a destra della scritta “View Controller Scene”, nella parte sinistra di Xcode, dov’è mostrata la struttura del view controller:

.....  
.....  
.....  
.....  
.....

**NOTA BENE:**

Questo documento è un estratto del libro

## **Creare applicazioni per iPhone e iPad con Swift**

La versione completa di quest'opera è disponibile:

Sul sito web dell'autore: <http://www.travagliante.com>

(pagina: <http://www.travagliante.com/creare-applicazioni-per-iphone-e-ipad-con-xcode-e-swift-la-guida/>)

e in tutte le più importanti librerie digitali e market on-line

.....  
.....

---

## **Capitolo 11 - Uso della Fotocamera e della Libreria Foto**

### **L'accesso alla Camera dell'iPhone o dell'iPad**

Nel capitolo 10 abbiamo visto come sia possibile e facile realizzare un'applicazione di una certa utilità come una To-Do list.

Un'app completa di questo tipo avrebbe dovuto avere anche delle funzionalità per il salvataggio degli elementi completati della lista e avrebbe dovuto prevedere la possibilità di creare più di una lista.

Ovviamente, l'implementazione di funzionalità di questo genere prescinde dagli scopi di una guida come questa, per cui nel capitolo precedente ci siamo concentrati soprattutto sull'impiego di elementi nuovi, come il Table View Controller e il Navigation Controller.

Adesso, vediamo come effettuare l'accesso ad uno degli elementi più interessanti di un iPhone o di un iPad: la fotocamera.

Su App Store possiamo trovare una grandissima quantità di applicazioni basate sull'impiego della fotocamera (ad esempio, app di svago con cui l'utente può scattare delle foto e, successivamente, applicare filtri e/o effetti interessanti e divertenti, app di produttività che emulano funzionalità di scanner, ecc.).

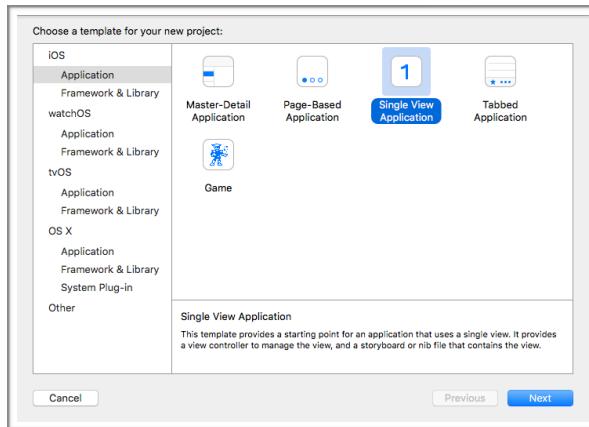
Generalmente, si tratta di applicazioni molto gradite agli utenti finali, perciò ritengo utile analizzarne le basi del loro funzionamento.

Quindi, realizziamo una semplice app che accede alla fotocamera del nostro iDevice. Più precisamente, sviluppiamo un'app che permette di mostrare una immagine catturata dalla fotocamera, oppure prelevata dalla libreria foto.

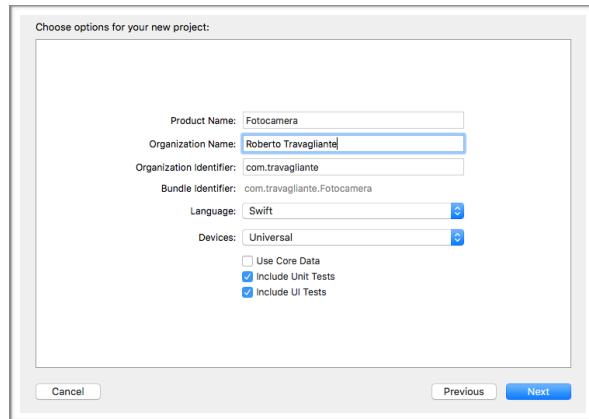
Lo sviluppo di questa app ci permetterà anche di analizzare il sistema dei protocolli e dei delegate, accessibile tramite Xcode e Swift, che ritengo un argomento importante se si desidera diventare un buon programmatore di applicazioni per iOS.

Come al solito, avviamo l'ambiente di sviluppo Xcode e diamo vita al nostro nuovo progetto, selezionando la voce di menu “File” -> “New” -> “Project...”.

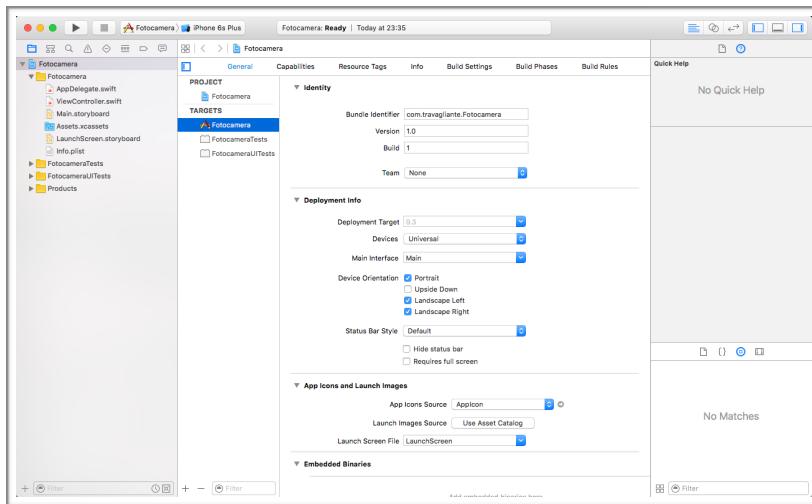
Dalla finestra di dialogo contenente le diverse tipologie di app, scegliamo la voce “Single View Application”:



Clicchiamo sul pulsante “Next” e nella schermata successiva selezioniamo il linguaggio Swift e un’app di tipo Universal, in grado cioè di funzionare sia su iPhone che su iPad. Denominiamo questo progetto “Fotocamera”:



Clicchiamo nuovamente sul pulsante “Next” e diamo inizio allo sviluppo della nostra app Fotocamera.



## La Storyboard e l’interfaccia utente

A questo punto, iniziamo a disegnare l’interfaccia utente per la nostra app.

Per gli scopi di questa guida, creeremo un’app basata su un’unica maschera contenente un elemento di tipo `UIImageView`, contenente l’immagine di volta in volta acquisita tramite la fotocamera o la libreria foto, oltre a due pulsanti (`UIButton`) che avranno le seguenti funzioni:

- “Fotocamera” - servirà per acquisire l’immagine mediante l’uso della fotocamera;
- “Libreria foto” - permetterà di aprire la libreria delle immagini contenute nel nostro iPhone o iPad e di sceglierne una da mostrare nella `UIImageView`.

.....  
.....  
.....  
.....  
.....  
.....  
.....

NOTA BENE:

Questo documento è un estratto del libro

## **Creare applicazioni per iPhone e iPad con Swift**

La versione completa di quest'opera è disponibile:

Sul sito web dell'autore: <http://www.travagliante.com>

(pagina: <http://www.travagliante.com/creare-applicazioni-per-iphone-e-ipad-con-xcode-e-swift-la-guida/>)

e in tutte le più importanti librerie digitali e market on-line

.....  
.....

---

## Tutte le guide di Roberto Travagliante

(per maggiori informazioni: <http://www.travagliante.com>)

### Creare Applicazioni per iPhone e iPad con Swift

La guida pratica per creare applicazioni di successo per iPhone e iPad, con Xcode e il linguaggio Swift, da zero!



### WordPress dalla A alla W

La guida completa per creare il tuo blog o il tuo sito web con WordPress, ottenendo il massimo!



## **Self Publishing**

Crea, Promuovi e Vendi il tuo e-book su 102 siti web! E diventa un autore di successo!



## **iPhone e iPad - Trucchi e Segreti di iOS 9**

La guida pratica con i migliori suggerimenti per essere subito “operativo” con iOS

