Swift – Stanford课程 – iOS11

第一节课

1. What will I learn in this course?

1. How to build cool apps?

\* Easy to build even very complex applications.

\* Result lives in your pocket or backpack!

\* Very easy to distribute your application through the AppStore.

\* Vibrant development community.

2. Real-life Object-Oriented Programming

\* The heart of Cocoa Touch is 100% object-oriented.

\* Application of MVC design model.

\* Many computer science concepts applied in a commercial development platform:

\* Databases, Graphics, Multimedia Multithreading, Animation, Network,and much, much more!

\* Numerous students have gone on to sell products on the AppStore.

2. Prerequisites

1. Prior Coursework

\* Object-Oriented Programming experience mandatory.

\* CS106A(or X) required & CS107 or CS108 or CS110 also (at a minimum) required.

\* (or equivalent for non-Stanford undergrads)

3. What's in iOS?

在iOS中教你什么？

\* Cocoa Touch

\* Media

\* Core Services

\* Core OS

1. 最底层Core OS 靠近硬件

2. 最顶层Cocoa Touch靠近用户

2. Core OS

\* OSX Kernel - Power Management

\* Mach 3.0 - Keychain Access

\* BSD - Certificates

\* Sockets - File System

\* Security - Bonjour

BSD 是 Unix 的变体，这里发生的事情基本上都在C语言中。因为 Unix主要是用C语言编写的，所以大部分内容都在C语言中。（这个不会教）

3. Core Services

\* Collections - Core Location

\* Address Book - Net Services

\* Networking - Threading

\* File Access - Preferences

\* SQLite - URL Utilities

这是一个面向对象的层级

4. Media

\* Core Audio - JPEG, PNG, TIFF

\* OpenAl - PDF

\* Audio Mixing - Quartz(2D)

\* Audio Recording - Core Animation

\* Video Playback - OpenGL ES

5. Cocoa Touch

\* Multi-Touch - Alerts

\* Core Motion - Web View

\* View Hierarchy - Map Kit

\* Localization - Image Picker

\* Controls - Camera

4. Demo

我们会在Demo中学习到如下的内容：

1. Concentration Game

\* All this stuff can be very abstract until you see it in action.

\* We'll start getting comfortable with Swift 4 and Xcode 9 by building something right away.

\* Two part demo starting today, finishing on Wednesday.

2. Today's topics in the demo

\* Creating a Project in Xcode 9, including building a UI and running in the iOS Simulator

\* Subclassing in Swift, including how to specify instance variables and methods

\* Connecting UI elements to invoke methods in our Swift code(actions)

print(outputting to the console using\() notation)

\* Connecting properties (instance variables) from our Swift code to the UI(outlets)

\* Accessing iOS documentation from our code

\* Automatically doing something every time a property's value changes

\* Array

\* Optionals

5. Coming Up

1. Today

\* Reading Assignment I (R1) assigned (due on Monday next week)

2. Wednesday

\* MVC(Model View Controller) design pattern

\* Concentration demo continued

\* Programming Assignment 1 (R1) assigned (mostly to recreate today's and Wednesday's lecture)

3. Friday

\* Tips and tricks for getting the most out of Xcode 9

\* Using the Debugger in Xcode 9

\* Hewlett 205, 11:30 - 12:20

## 制作游戏Demo

需求：卡片匹配游戏，2个卡片配对成功则消失，不成功则保留直到配对成功。

步骤：

1. 在SB中创建一个UIButton，连线到ViewController类中。方法名为 @IBAction func touchCard(\_ sender: UIButton)

func flipCard(withEmoji emoji: String, on button: UIButton) {...}

/// 点击卡片

@IBAction func touchCard(\_ sender: UIButton) {

flipCard(withEmoji: "👻", on: sender)

}

/// 翻转卡片

func flipCard(withEmoji emoji: String, on button: UIButton) {

if button.currentTitle == emoji {

button.setTitle("", for: UIControl.State.normal)

button.backgroundColor = UIColor.orange

}else {

button.setTitle(emoji, for: UIControl.State.normal)

button.backgroundColor = UIColor.white

}

}

// 创建一个变量，用来跟踪卡片的翻转次数（点击次数）

var flipCount: Int = 0

// Swift 是一种具有强类型推断的语言，它可以为你猜出类型。

// 所以可以不用写 Int

var flipCount = 0

func flipCard(withEmoji emoji: String, on button: UIButton) {

// 使用变量来跟踪卡片的点击次数

flipCount += 1

}

// 创建一个UILabel，显示翻转计数

@IBOutlet weak var flipCountLabel: UILabel!

func flipCard(withEmoji emoji: String, on button: UIButton) {

// 使用变量来跟踪卡片的点击次数

flipCount += 1

// 给Label 添加卡片的点击次数

flipCountLabel.text = "Flips:\(flipCount)"

}

**优化：**

现在只有一张卡片，如果有2张，10张，每次都写一个卡片点击方法，这代码会非常臃肿。

我们可以使用swift中的didSet方法。

* didSet: 每次设置该变量的时候，都会执行didSet中的内容代码。

var flipCount = 0 {

didSet {

flipCountLabel.text = "Flips: \(flipCount)"

}

}

创建一个var变量，这个变量里面保存着所有的UIButton。SB连线到VC的时候，选择 Outlet Collection.

@IBOutlet var cardButtons:[UIButton]!

func touchCard(\_ sender: UIButton) {

flipCount += 1

var cardNumber = cardButtons.firstIndex(of: sender)

print("cardNumber = \(cardNumber)")

}

上面的代码写完后，会出现错误提示

warning 1:

variable 'cardNumber' was never mutated; consider changing to 'let'

warning 2:

String interpolation produces a debug description for an optional value; did you mean to make this

改进：

func touchCard(\_ sender: UIButton) {

flipCount += 1

if var cardNumber = cardButtons.firstIndex(of: sender) {

print("cardNumber = \(cardNumber)")

}else {....}

}

var emojiChoices: Array<String> = ["👻","🎃","🎃","👻",]

func touchCard(\_ sender: UIButton) {

flipCount += 1

if var cardNumber = cardButtons.firstIndex(of: sender) {

// 新加内容

flipCard(withEmoji: emojiChoices[cardNumber], on: sender)

}else {....}

}

整体代码：

import UIKit

class ViewController: UIViewController {

var emojiChoices: Array<String> = ["👻","🎃","🎃","👻",]

/// 保存所有的UIButton

@IBOutlet var cardButtons: [UIButton]!

/// 创建一个变量，用来跟踪卡片的翻转计数

var flipCount = 0 {

didSet {

// 给Label显示文字

flipCountLabel.text = "点击次数:\(flipCount)"

}

}

/// 显示翻转计数的Label

@IBOutlet weak var flipCountLabel: UILabel!

/// 点击卡片

@IBAction func touchCard(\_ sender: UIButton) {

flipCount += 1

print("点击\(flipCount)")

if let cardNumber = cardButtons.firstIndex(of: sender) {

flipCard(withEmoji: emojiChoices[cardNumber], on: sender)

} else {

}

}

/// 翻转卡片

func flipCard(withEmoji emoji: String, on button: UIButton) {

if button.currentTitle == emoji {

button.setTitle("", for: UIControl.State.normal)

button.backgroundColor = UIColor.orange

}else {

button.setTitle(emoji, for: UIControl.State.normal)

button.backgroundColor = UIColor.white

}

}

}

第二节课

MVC

一. 前言

这节课前15分钟讲 一种设计模式MVC, 之后我们要使用Model-View-Controller 到我们的应用程序中。

1. MVC

\* Object-Oriented Design Pattern

2. Continuation of Concentration Demo

\* Use MVC to make our Concentration game a lot smarter

\* Creating our own data Structures (Concentration and Card)

\* Initialization

二、MVC

1. Model - View - Controller 是什么？

我们将系统中的所有对象分成两部分，三个阵营。

Model：模型

这是一组独立于UI的对象

Controller：控制器

How your Model is presented to the user( UI logic)

控制器是它在屏幕上显示的方式。

View：视图

它是控制器的仆人，它有很多通用的UI元素，如UI按钮。

2. MVC的一些规则

1. 控制器可以和Model交谈

2. 控制器可以和View交谈

3. Model 不能跟 View 交谈，View 也不能跟 Model 交谈

原因：

1. 模型是独立于UI的，而视图仅仅只有UI的东西，所以绝对没有UI控制模型的东西。

2. UI是一个界面的封装，不需要知道任何东西。

4. View 可以跟Controller 通信

3. MVC可以有很多个。

一个app中，可以有很多个MVC，一个界面可能就是一个MVC，每个MVC可能会相互关联

4. 在这个Demo中，我们要创建Controller，然后将model跟controller关联起来。

Struct 和 class 的区别

1. struct 没有继承

代码：

创建一个model类，名称是 Concentration.swift

当我们创建一个 class 的时候，需要想一下 公共API.

* API ： 应用程序编程接口。 Application Programing Interface
* 一个类中会有方法和实例变量
* 公共API 是所有实例变量和方法，允许其他类调用

基础知识

一、可选类型（Optionals）

（一）、基础知识

1. 作用：用于处理值缺失的情况。

2. 解释：可选表示“那儿有一个值，并且它等于X”或者“那儿没有值”。

3. 声明

var optionalInteger: Int?

var optionalInteger: Optional<Int>

4. Optional 是一个含有两种情况的枚举，用来表示可能有或可能没有值。

None

Some（T）

5. 声明可选整数数组，应该写成（Int[]）? , 写成 Int[] 会报错。

6. 当你声明一个可选变量或可选属性的时候没有提供初始值，它的值默认为nil。

7. 强制解析：如果一个可选类型的实例包含一个值，需要用后缀操作符！来访问这个值。

optionalInteger!

使用操作符！去获取，当值为nil的时候，可选变量会报错。

可以用可选链接和可选绑定选择性执行可选表达式上的操作。如果值为nil，任何操作都不会执行，也不会有运行报错。

例如： var myString:String? = nil

if myString != nil {

print(myString)

}else{

print("字符串为 nil")

}

// 执行结果

字符串为nil

8：自动解析： 可以在声明可选变量时使用感叹号(!)替换问号(?). 这样可选变量在使用时就不需要在加上一个感叹号(!)来取值，它会自动解析。

var myString: String!

myString = "Hello, Swift!"

if myString != nil {

print(myString)

}else{

print("字符串为 nil")

}

// 执行结果

Hello swift!

9. 使用可选绑定来判断可选类型是否包含值，如果包含就把值赋给一个临时常量或者变量。可选绑定可以用在if和while语句中来对可选类型的值进行判断并把值赋给一个常量或者变量。

if let constantName = someOptional {

statements

}

注意：可选类型类似于Objective-C中指针的nil值，但是nil只对类（class）有用，而可选类型对所有的类型都可用，并且更安全。

（二）、文章

[Swift 中的Optional详解](https://www.jianshu.com/p/8b622333d6db)

https://blog.csdn.net/Z1591090/article/details/120613913

Swift之深入解析可选类型Optional的底层原理