GENERAL ASSEMBLY

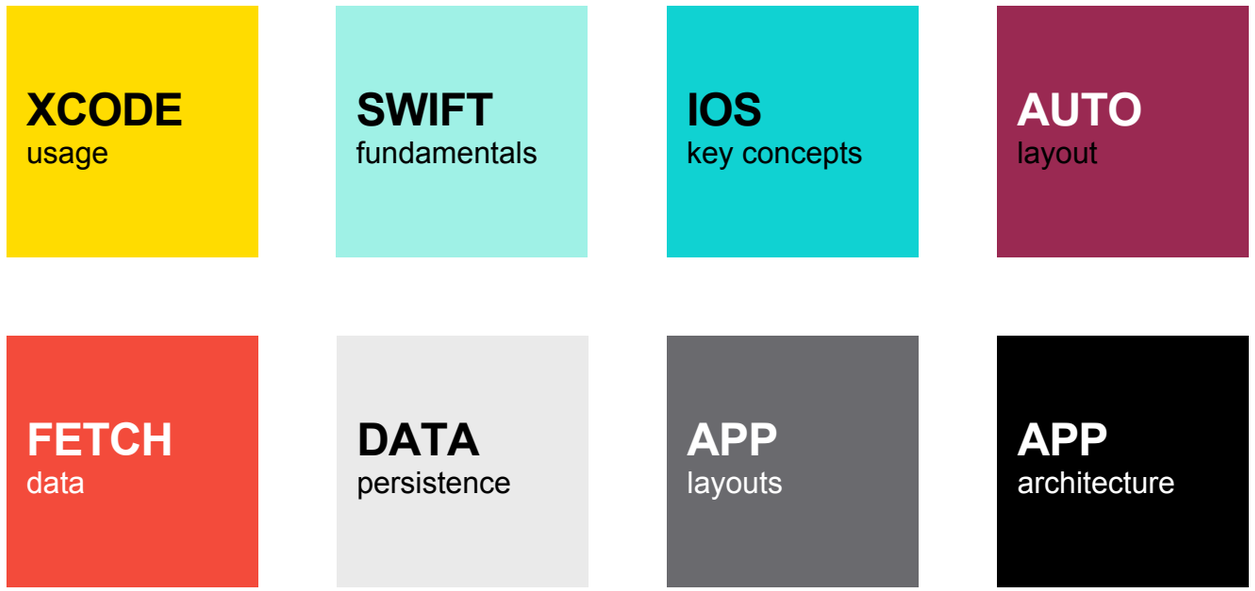
INTRO TO SWIFT

Giorgio Natili @giorgionatili, Engineering Mentor and Technical Lead

AGENDA

- What’s New in XCODE 6
- Swift Language
- Key Concepts
- iOS Auto Layout
- User Interaction
- Remote and Local Data
- Core Data Basic
- UITableView and Custom Cells

TAKEAWAY


XCODE
usage
SWIFT
fundamentals
IOS
key concepts
AUTO
layout
FETCH
data
DATA
persistence
APP
layouts
APP
architecture

HELLO!

GIORGIO NATILI

- Technical Leader Agile Coach
- Front-end Developer
- Mobile Developer (Cordova, iOS, Android)
- Technology Enthusiast
- Open Source Fellow
- Mentor @ Airpair.com

MOBILE TEA

mobiletea

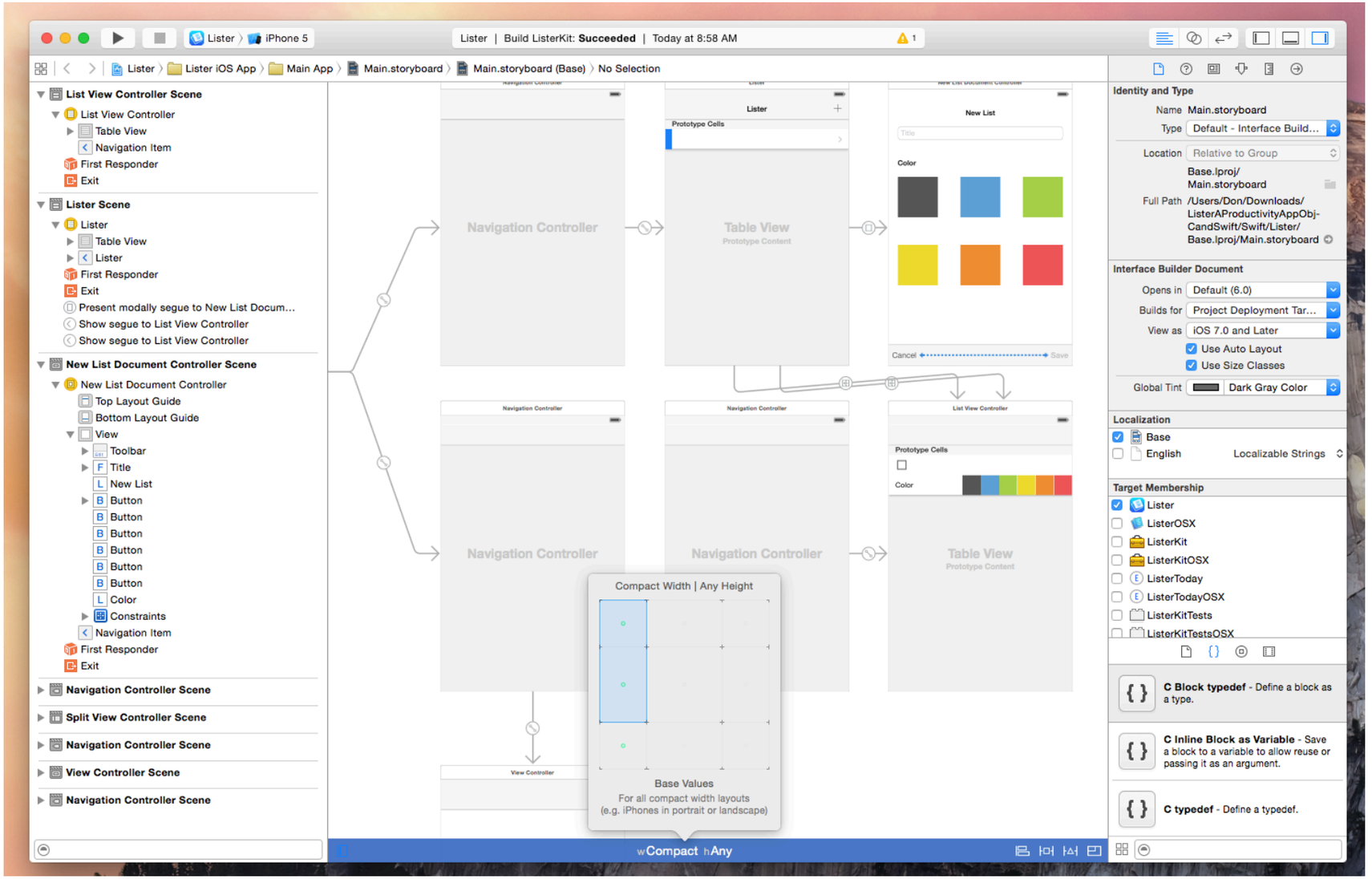
http://www.meetup.com/mobiletea

XCODE 6.X

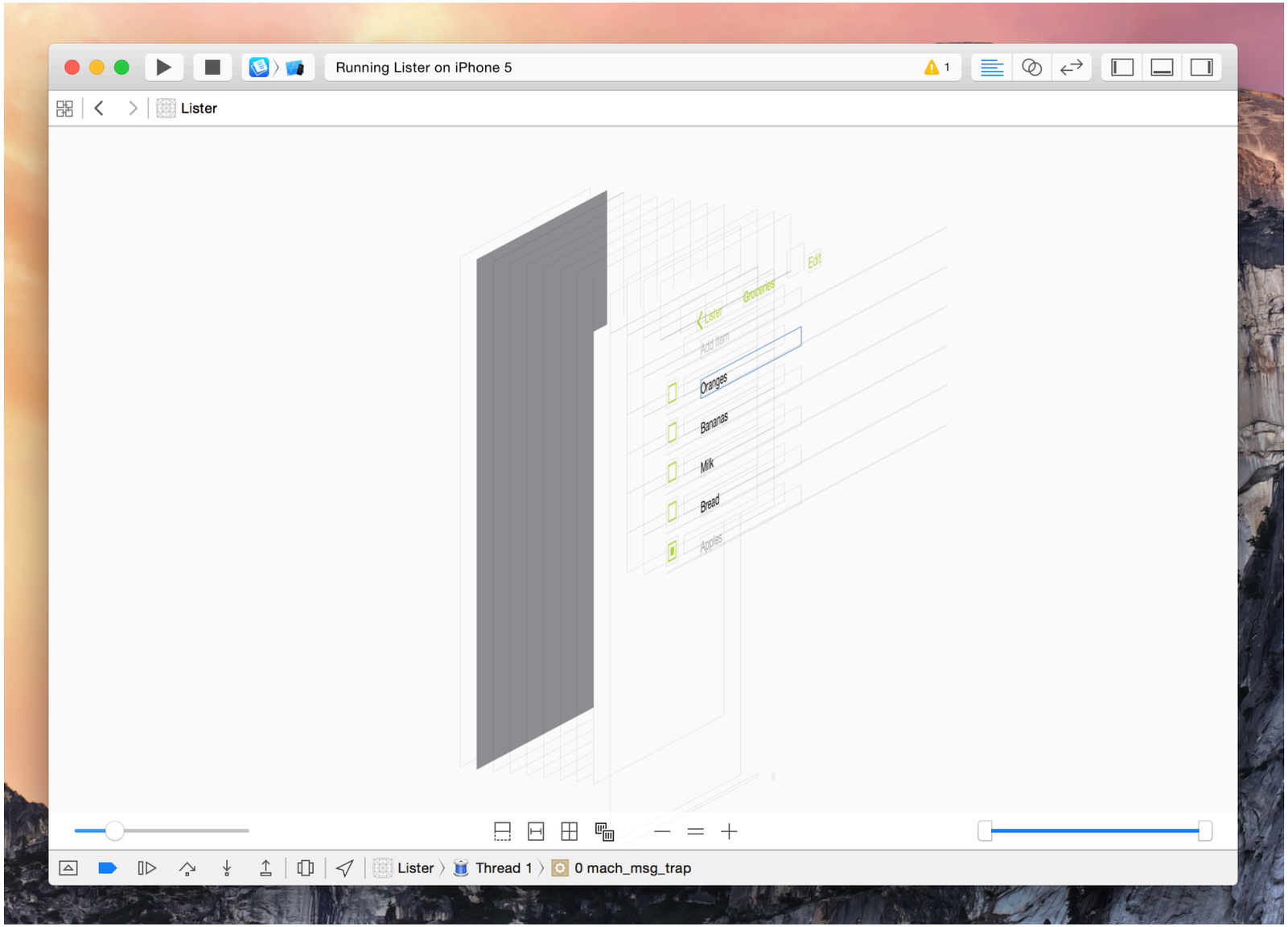
FEATURES

- Live rendering
- Visual view debugging
- Playground
- Swift
- Asynchronous code testing
- iOS simulator

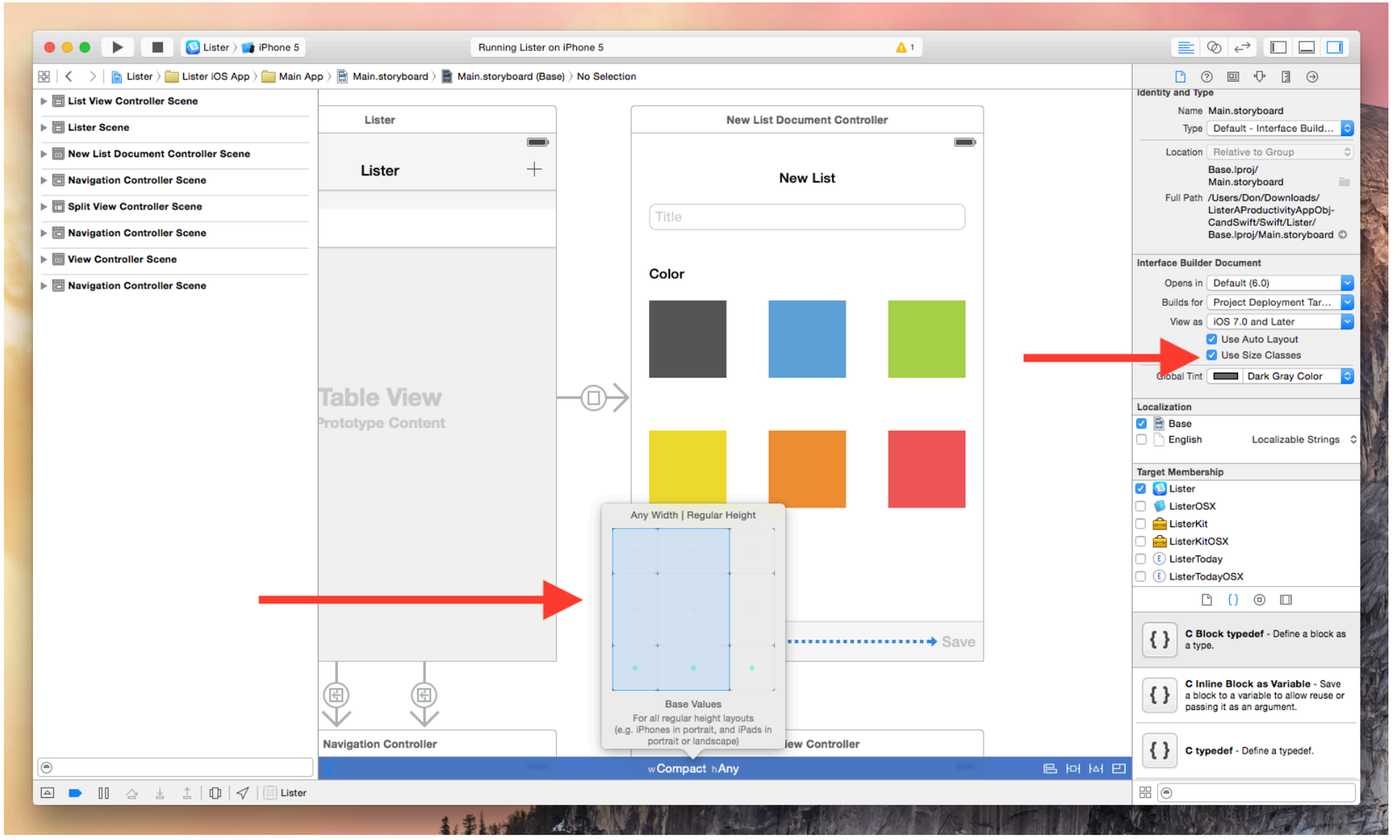
VISUAL DESIGN



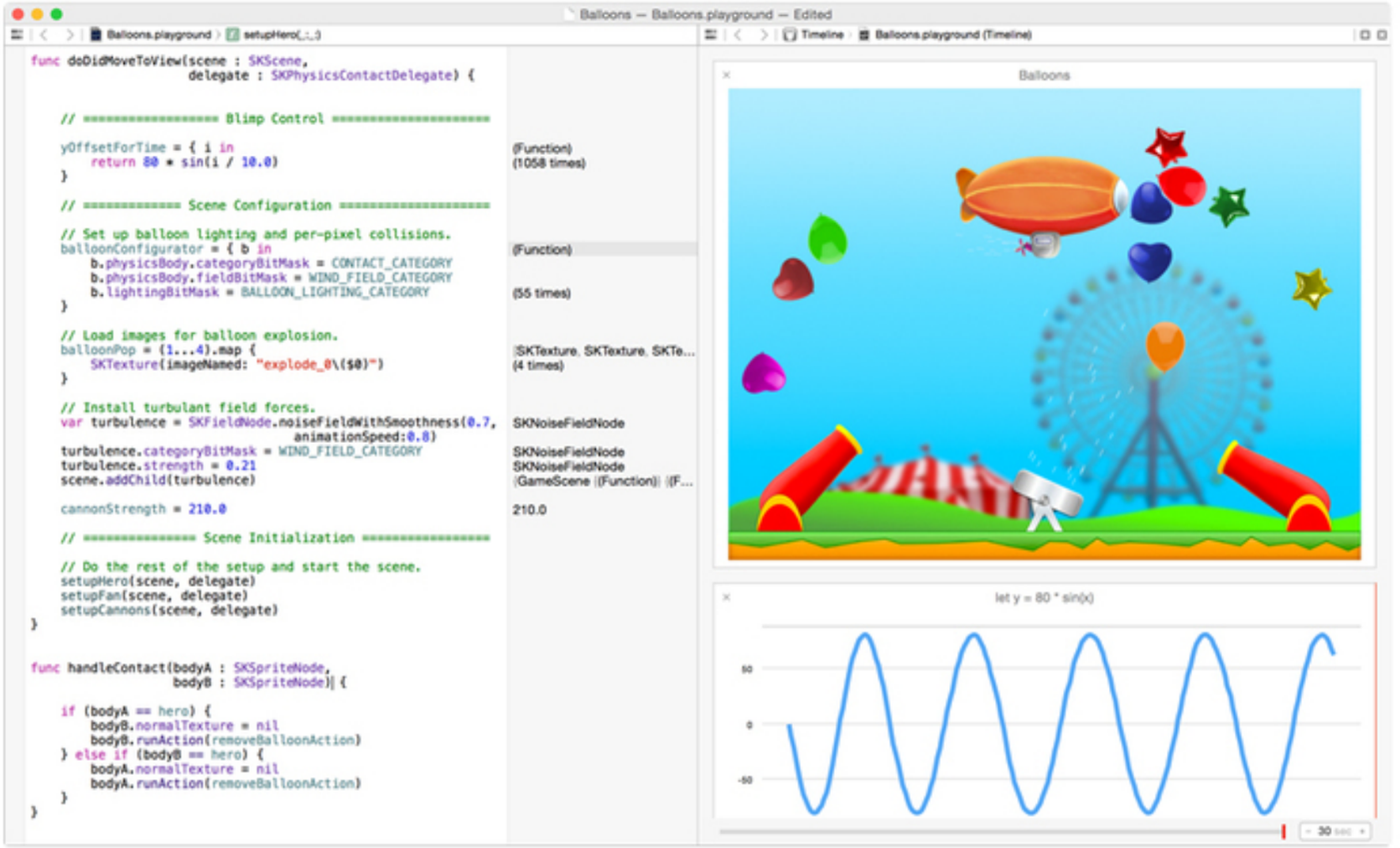
LIVE RENDERING



MULTI SCREEN DESIGN



PLAYGROUND


Balloons — Balloons.playground — Edited
Balloons.playground
setupHero(_:_:)
Timeline
Balloons.playground (Timeline)
Balloons
let y = 80 * sin(x)
50
0
-50
func doDidMoveToView(scene : SKScene,
delegate : SKPhysicsContactDelegate) {
// ================== Blimp Control ====================
yOffsetForTime = { i in
return 80 * sin(i / 10.0)
}
// ============= Scene Configuration ====================
// Set up balloon lighting and per-pixel collisions.
balloonConfigurator = { b in
b.physicsBody.categoryBitMask = CONTACT_CATEGORY
b.physicsBody.fieldBitMask = WIND_FIELD_CATEGORY
b.lightingBitMask = BALLOON_LIGHTING_CATEGORY
}
// Load images for balloon explosion.
balloonPop = (1...4).map {
SKTexture(imageNamed: "explode_0\($0)")
}
// Install turbulant field forces.
var turbulence = SKFieldNode.noiseFieldWithSmoothness(0.7,
animationSpeed:0.8)
turbulence.categoryBitMask = WIND_FIELD_CATEGORY
turbulence.strength = 0.21
scene.addChild(turbulence)
cannonStrength = 210.0
// ============== Scene Initialization =================
// Do the rest of the setup and start the scene.
setupHero(scene, delegate)
setupFan(scene, delegate)
setupCannons(scene, delegate)
}
func handleContact(bodyA : SKSpriteNode,
bodyB : SKSpriteNode) {
if (bodyA == hero) {
bodyB.normalTexture = nil
bodyB.runAction(removeBalloonAction)
} else if (bodyB == hero) {
bodyA.normalTexture = nil
bodyA.runAction(removeBalloonAction)
}
}
(Function)
(1058 times)
(Function)
(55 times)
[SKTexture, SKTexture, SKTe...
(4 times)
SKNoiseFieldNode
SKNoiseFieldNode
SKNoiseFieldNode
[GameScene, (Function), (F...
210.0

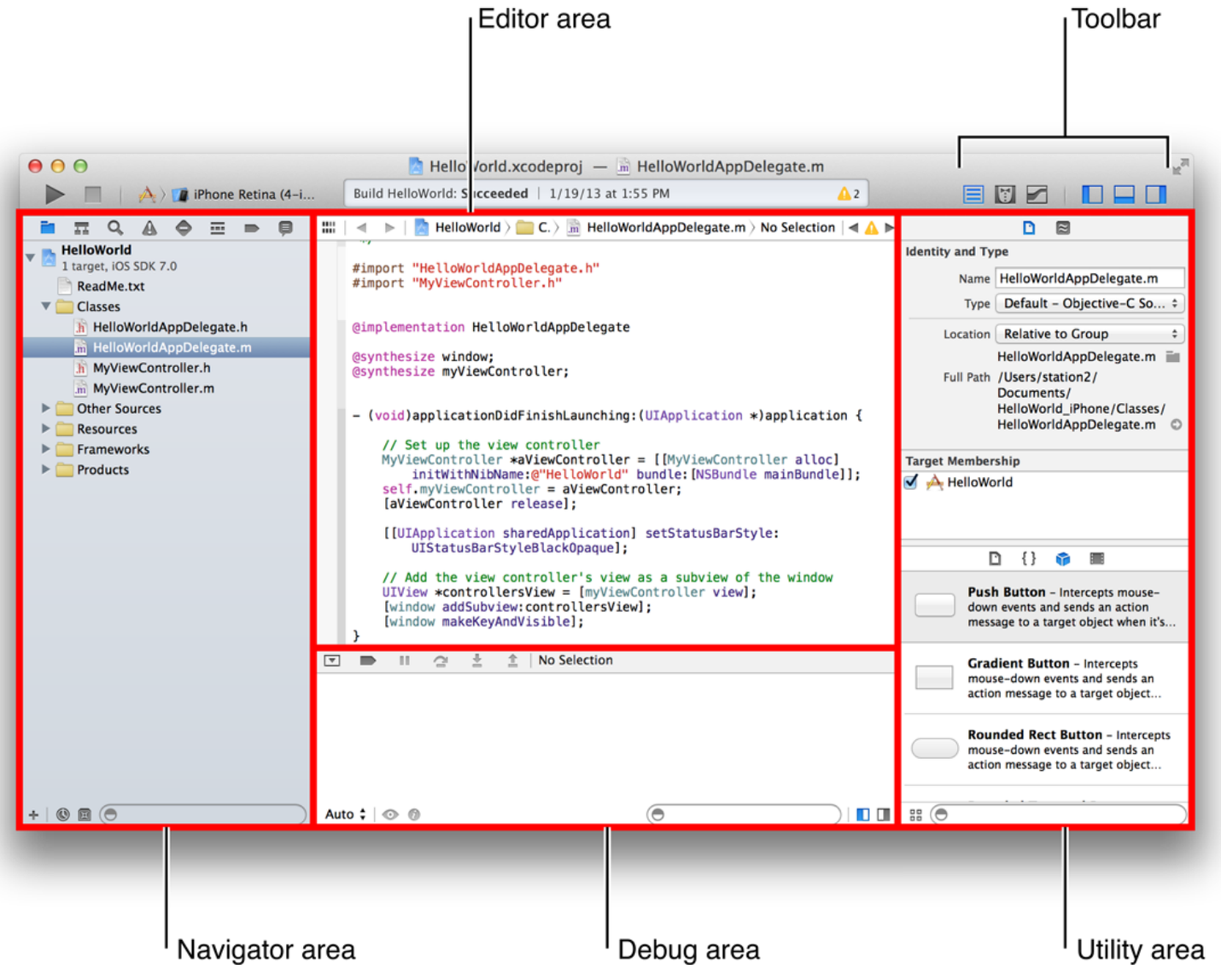
IOS SIMULATOR

- Improved SDK maintaining
- More devices
- Resizable devices

ASYNCHRONOUS CODE TESTING

- XCTest API enhancements
- Network calls
- File IO

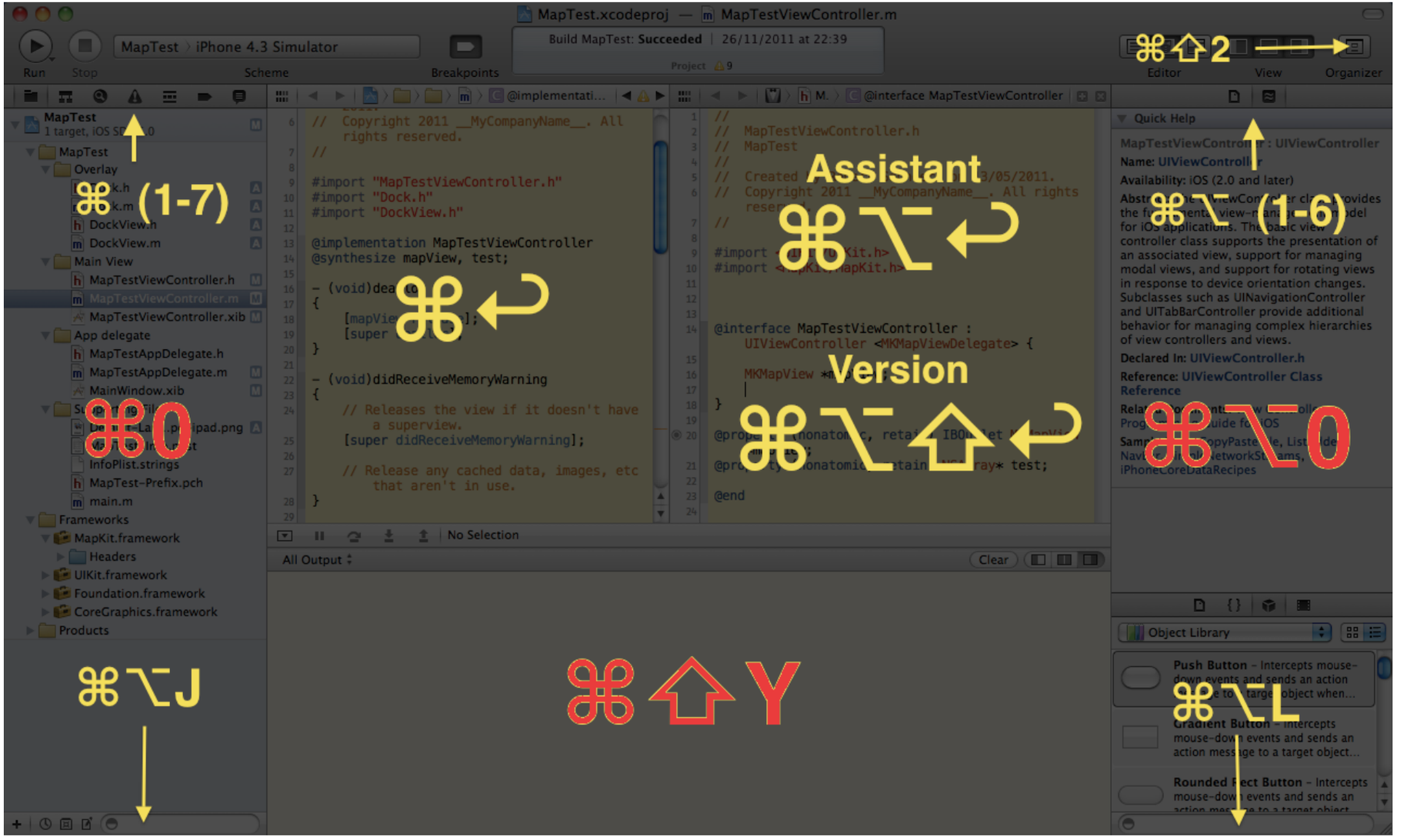
XCODE AT A GLANCE



SHORTCUTS – COMMON KEYS

- ⌘ = Command
- ⇧ = Shift
- ⌥ = Option/Alt
- ^ = Control
- ⇠⇢ = Left/Right Arrow Keys
- ⇡⇣ = Up/Down Arrow Keys

WORKSPACE AND SHORTCUTS



http://www.rsaunders.co.uk/2011/11/xcode-shortcuts.html

SHORTCUTS – BOOST PRODUCTIVITY

⇧ + ⌘ + O	*Open a file*
^ + ⌘ + ⇡ OR ⇣	*Switch between .m and .h files*
⌘ + click on a variable/type	*Go to definition*
^ + ⌘ + ⇠ OR ⇢	*Go forward / Go back*
^ + ⌘ + E	*Edit all in scope*
⌘ + T	*Open a new tab*
⌘ + W	*Close the current tab*
⇧ + ⌘ + [**or**]	*Go forward / Go back in tabs*

SHORTCUTS – DEBUGGING

F6	*Step over*
F7	*Step into*
⌘ + \	*Add/remove breakpoint*
⌘ + Y	*Disable/enable all breakpoints*
^ + ⌘ + Y	*Pause/Play the debugger*

SWIFT BASICS

DATA TYPES – BASICS

- `Char`
- `Int`
- `Float`
- `Double`

NAMED AND COMPOUND TYPES

- A named type can have a name assigned

  Structures (e.g. Number and Strings), Classes, Enumeration, Protocols

- A compound type has no name

  Tuples and functions

VARIABLES AND CONSTANTS

In Swift you can define a variable or a constant using different key words:

- `var`
- `let`

The general syntax to specify the type of the value stored in the var/const is:

```
var/let identifier: <Type>
```

TYPE INFERENCE

The variable or constant *types* are defined from the compiler

```
var i:Int = 10
```

is the same of

```
var i = 10
```

ARITHMETIC EXPRESSIONS

- Most of the commons defined in other modern languages: `+`, `-`, `/`, `=`, `%`, `+=`, `-=`, `*=`, `%=`
- Pre and post increment/decrement operators: `++x`, `x++`, `--x`, `x--`

```
var x = 0
var y = x++
```

```
var y = ++x
```

BOOLEAN EXPRESSIONS

- Relational operations as in other modern languages: ==, <, >, <=, =>, !=
- Ternary operator: ?

```
[condition] ? [execute if true] : [execute if false]
```

CONDITIONAL STATEMENTS

```
if <condition>{

}

switch value {
    case something:
    case somethingElse:
    default:
}
```

CONTROL TRANSFER STATEMENTS

- `return`: stops the execution and return a value
- `falltrough`, force a C style behavior and also subsequent cases are evaluated
- `continue`, tells a loop to stop what it is doing and start again at the beginning of the next iteration
- `break`, ends execution of an entire control flow statement immediately

LOOPS

Supports the most common loops: for, while, do…while, etc. but has couple of nice hints for ranges and parameters:

```
for i in 0…9{
    println("i = \(i) ”)
}

for _ in 0..<10{
    println("I don’t care about i")
}
```

COLLECTION TYPES – ARRAY

- An array stores an ordered list of values of the same type
- There is no more disitinction between `NSArray` and `NSMutableArray`
- An array can be instantiated using the literal syntax or the array type syntax

```
var languages: [String] = ["swift", "objc"]
```

is the same of

```
var languages = ["swift", "objc"]
```

COLLECTION TYPES – ARRAY METHODS

- `count`
- `isEmpty`
- `append`
- `insert(that, atIndex:)`
- `removeAtIndex(index)`

COLLECTION TYPES – DICTIONARY

A dictionary it's a collection that stores key-value pairs with all the keys having the same type and all values having the same type as well.

```
var states = ["NY": "New York", "MA": "Massachusett"]
```

- `count`
- `isEmpty`
- `updateValue(val, forKey:)`
- `removeValueForKey`

FUNCTIONS

- `func` is followed by the `identifier`
- `identifier`, is followed by a parameter list enclosed in parentheses
- `->`, optionally specifies the return type of the function

```
func sayHello(name:String)->String{
    return "Hello " + name
}
```

FUNCTIONS – CONSTANT AND VARIABLE PARAMETERS

- By default functions parameters are constant
- Optionally can be declared as variables `var`

```
func hello(var message:String, name:String)->String{

    message += name
    return message

}
```

FUNCTIONS – INOUT PARAMETERS

- When a function is used to modify an existing value it's possible to pass it as an `inout` parameter

```
var value
func updateValue(inout data:T){
    // Do something
}
updateArray(&value)
```

LAB 1

Calculate the average number of a series of numbers stored in an Array using a function (it’s assumed you will use the Playground).

CLOSURES

Closures are expressions that resemble “anonymous” or unnamed functions; a closure is a self-contained blocks of functionality that can be passed around in your code.

```
let helloFromClosure = {(name:String) -> String in
    return "Hello " + name + " from your closure! "
}
```

CLOSURES + ARRAY

Ordering an Array with a closure make the code more compact and easy to read

```
let animals = ["Dog", "Cat", "Fish", "Worm"]
let sortedStrings = animals.sorted(
                {(one:String, two:String) -> Bool in
                    return one < two
})
```

LAB 2

Find the sum of all even numbers between 1 and 10 contained in an Array using a function or a closure.

CLASSES

- The syntax is similar to the one of the most common languages
- A class can contains method and `stored` or `computed` properties

```
class Person{
        var name:String = "Giorgio"
        var computedName:String {
                get { return name + " " + rand().description}
        }
}
```

CLASSES – INITIALIZERS

- Each class must have an initializer
- Strictly speaking the initializer is where the class member will be initialized (partially not true)
- An initializer accept parameters like a function
- A class can have multiple initializers that differ in their signature

CLASSES – INITIALIZERS

```
class Animal{
    init(){ // Some code }
}

class Animal{
    init(specie:String){
        println("I am a \(specie)")
    }
    init(specie:String, gender:String){
        println("I am a \(specie), my gender is \(gender)")
    }
}
```

CLASSES – MEMBERS SIGNATURE

- By default all the members have an `internal` access level
- Other optional access levels are `public`, `private`, `final` and `static`
- The access level of a custom type affect all the members of that type

THE INTERNAL LEVEL

It means internal to a product, a product is an app or a framework or any other distributable compiled item.

PROTOCOLS

- A protocol is a specification that list the properties and methods an implementer has to support
- A property is specified declaring a variable followed with a type and then either `{get}` or `{get set}`
- A method is specified using the `func` keyword followed by the function signature

```
protocol DailyGreting{
    var userName:String {get set}
    func welcome() -> String
}
```

LAB 3

Imagine to work in the finance industry and define a protocol to ensure that all the type of bank account are able to print out the full name of the owner, the balance and the account id. Once the protocol is ready define a an account class that implement the protocol.

STRUCTURES

A `struct` is a complex data type declaration that defines a physically grouped list of variables to be placed under one name in a block of memory. (*wikipedia*)

`struct` are value types, it means if you copy the instance of the structure to another variable it just copied to the variable

`class` are reference type, it means if you assign a instance of the class to another variable, it will hold only the reference of the instance not copy

STRUCTURES – MUTATING METHODS

A `struct` can have properties, methods, initializers and can be extended; a method that changes the value of a property is marked as `mutating`

```
struct Account{
    var accountId:Int
    var amount:UInt8 = 0
    init(id:Int){
        accountId = id
    }
    mutating func manageAmount(value:UInt8){ amount += value}
}
```

ENUMERATIONS

An enumeration is a common type for encapsulating a group of related values which can be accessed in a type-safe way within the code

```
enum AccountType{
    case Savings
    case Checking
}
```

ENUMERATIONS – WITH DATA TYPES

```
enum AccountType:String{
    case Savings = "savings"
    case Checking = "checking"
}

enum AccountID{
    case inString (String)
    case inNumber (Int)
}
```

SWIFT OPTIONALS

- Using optionals it's possible to assign a `nil` value to a variable
- An optional is defined using the `?` sign

```
var age:Int? = nil
```

- Optional is an enumeration defined in the Swift language protocol

```
 enum Optional : NilLiteralConvertible {
   case None
   case Some(T)
}
```

SWIFT OPTIONALS – UNWRAPPING

- An optional value needs to be unwrapped before using it, you can force the unwrap with the ! sign
- It’s safer to unwrap an optional using a kind of bind technique

```
if let value = possibleValue {

    // Something happens

}
```

FUNC OPTIONAL RETURN VALUES

- It’s possible that a function doesn’t return a value, an optional can be used also in the function return type

```
func maxValue(nums: [Int]) -> Int?{

    return nums.reduce(Int.min, { max($0, $1) })

}
```

TUPLES

Tuples enable you to create and pass around groupings of values

```
let http404Error = (404, "Not Found")
// http404Error.0 or http404Error.1
```

LAB 4

Implement a function able to return a tuple containing the max, the min and the average value of a sequence of numbers contained into an Array.

WE LOVE SWIFT!

HELLO WORLD!

HELLO WORLD APP

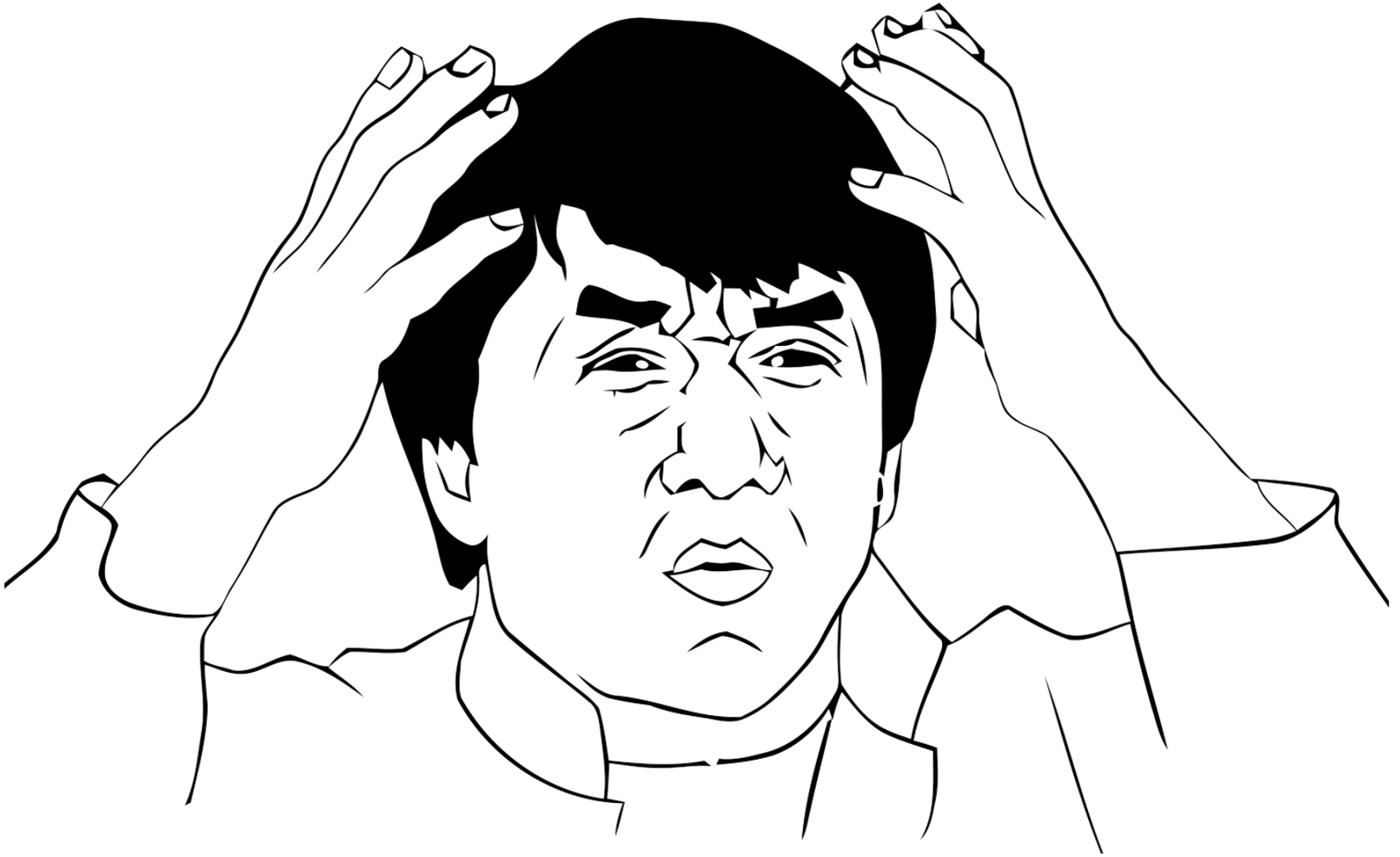
- Open XCODE
- Create a single view app
- Drag a button to the main view in the storyboard
- Right click on the button and create a binding for the touch up inside event
- Open an alert from the event handler

```
let alertController = UIAlertController(title: "Welcome to My First App", message:
"Hello World", preferredStyle: UIAlertControllerStyle.Alert)


alertController.addAction(UIAlertAction(title: "OK", style:
UIAlertActionStyle.Default, handler: nil))


self.presentViewController(alertController, animated: true, completion: nil)
```

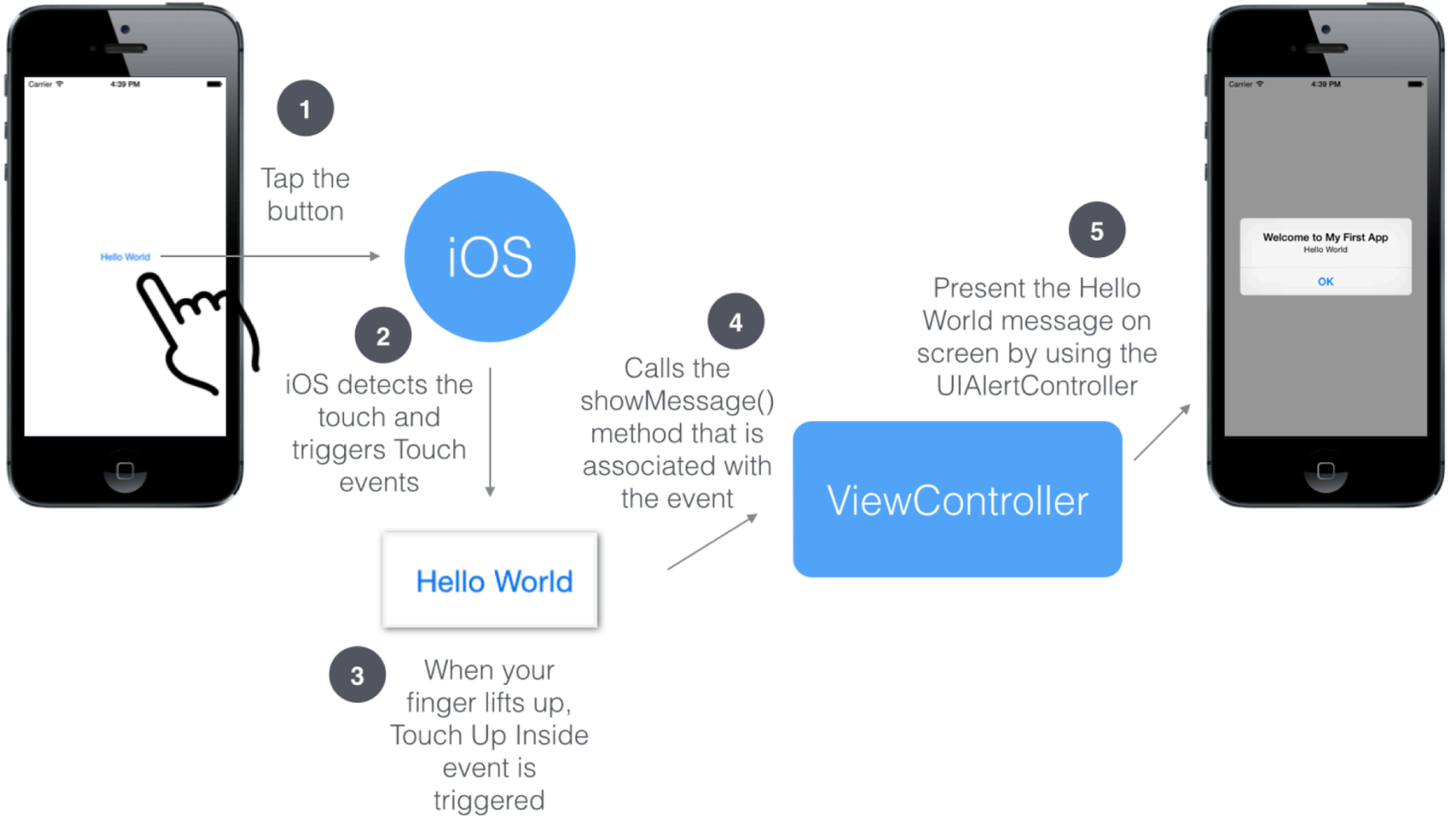
WHAT'S HAPPENED?



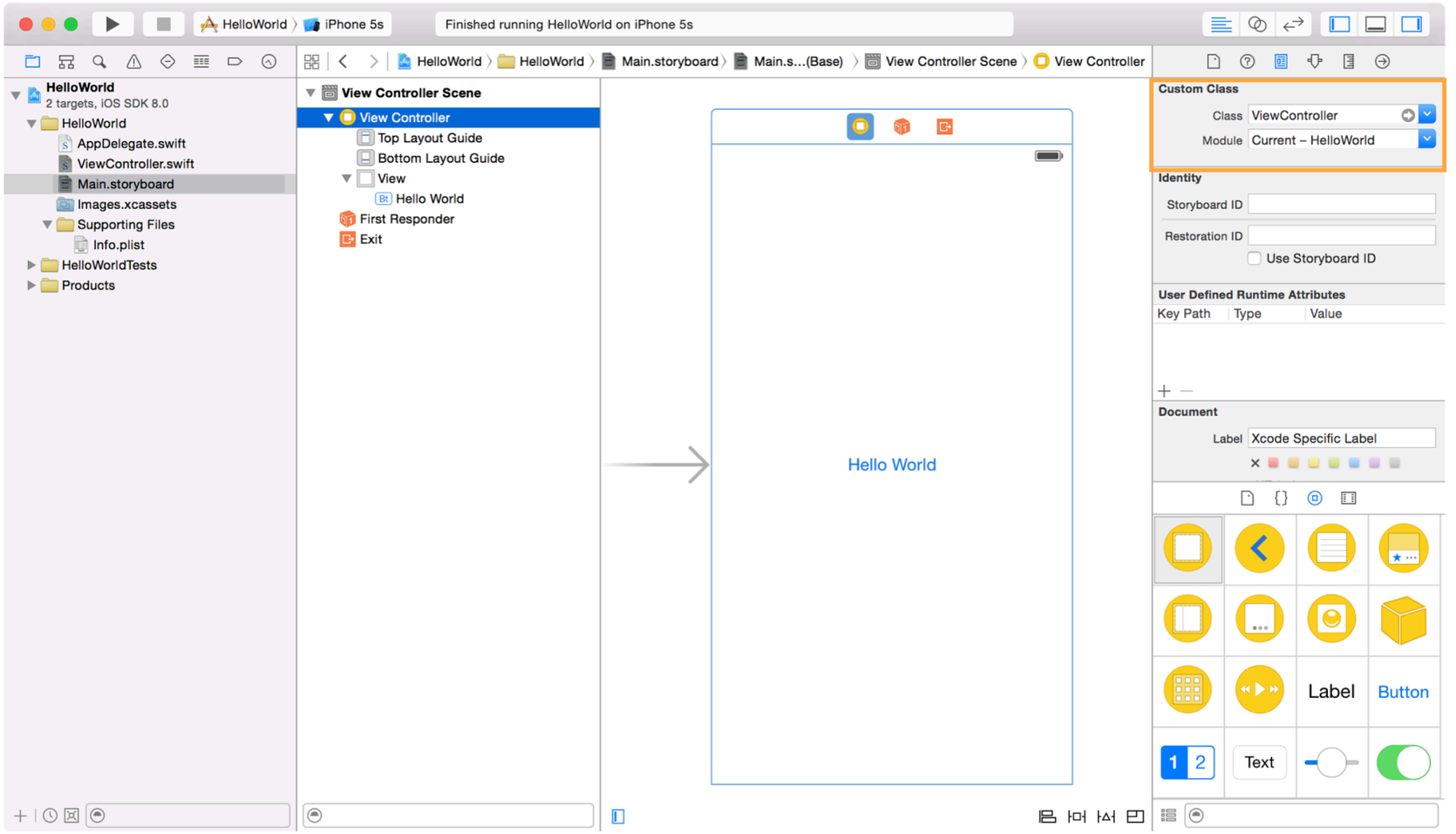
BASIC CONCEPTS

- The user interface in storyboard is the interface, while the code is the implementation
- The user interface elements (e.g. button) communicate with the code via messages
- The `@IBAction` keyword allows to connect your source code to user interface objects in Interface Builder

UNDER THE HOOD



UI AND VIEW CONTROLLERS



OWNER, RESPONDER AND EXIT

- The *File Owner* is an instantiated, runtime object that owns the contents of your .xib and its outlets/actions when the .xib is loaded
- The *First Responder* is simply the first object in the responder chain that can respond to events
- The *Exit* is the method to be executed when .xib or the View Controller are unloaded from the view

LAB 5

Create the Hello World application and add another View Controller to the storyboard; the view controller should be rendered when a button is touched and should render some text.

AUTO LAYOUT

DIFFERENT PLATFORMS BUT SAME LAYOUT ISSUES

UI DESIGN, RETINA AND 3X

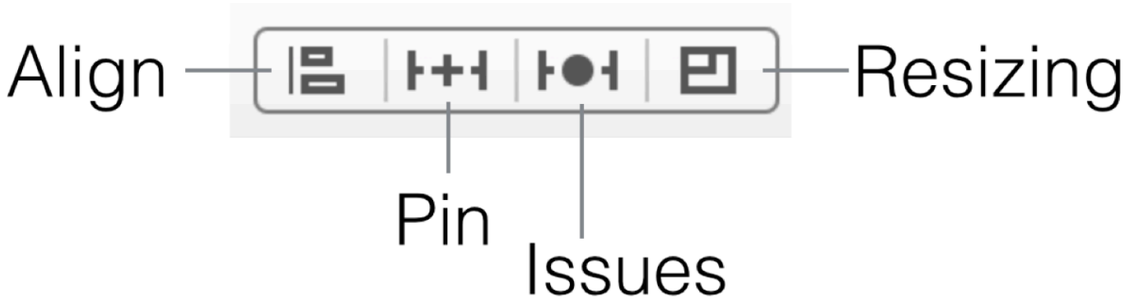
- In the storyboard or in a .xib file the UI element are positioned using a point based coordinate system
- Until retina display a point was the equivalent of a pixel
- With retina displays resolution doubled and a point corresponds to two pixels
- iPhone 6 Plus introduced another complication, a point now corresponds to three pixels

AUTO LAYOUT

- Auto Layout is a constraint-based layout system
- It allows developers to create an adaptive UI that responds appropriately to changes in screen size and device orientation
- In XCODE 6.1 allows to have a live preview in the storyboard itself

DEFINING CONSTRAINTS

- Align, create alignment constraints, such as aligning the left edges of two views
- Pin, create spacing constraints, such as defining the width of a UI control
- Issues, resolve layout issues
- Resizing , specify how resizing affects constraints



LAB 6

Open again the Hello World app and using Auto Layout make the UI elements rendering properly on the iPhone 4s, iPhone 5s and iPhone 6 Simulators.

TABLE VIEW

TABLE VIEW LAYOUT

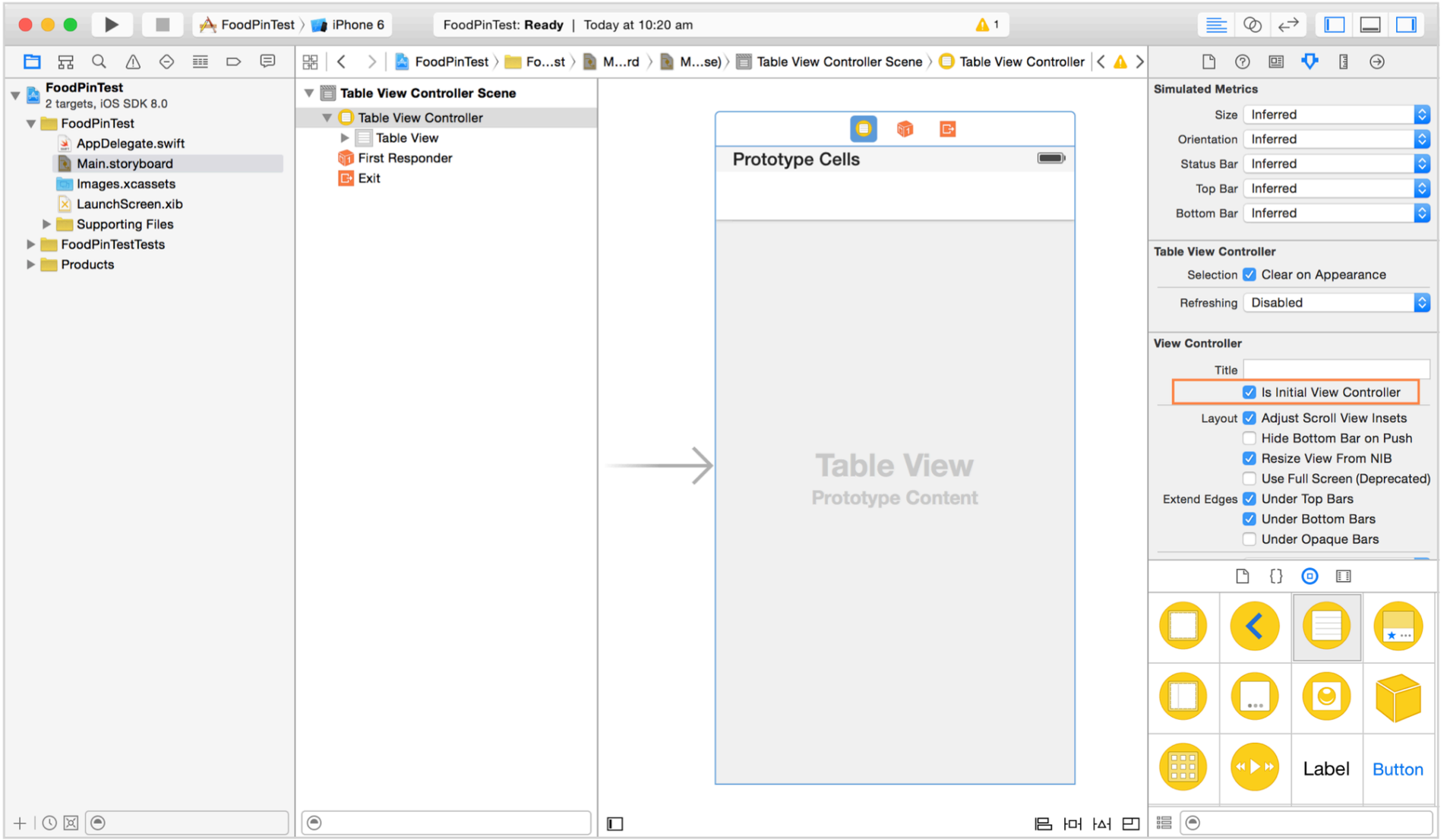


TABLE VIEW APPS

FETCH + PERSIST DATA

REMOTE DATA – JSON

- `NSURL`, represents a URL that can potentially contain the location of a resource on a remote server
- `NSURLSession`, provide an API for downloading content via HTTP
- `NSURLSession.dataTaskWithURL`, creates an HTTP GET request for the specified URL
- `NSJSONSerialization`, convert JSON to Foundation objects and viceversa

XCPSETEXECUTIONSHOULDCONTINUEINDEFINITELY
It forces the playground to keep continuing executing the code after the end of the code block.

LAB 7

Using the `NSURL`, `NSURLSession` and `NSJSONSerialization` APIs load a remote JSON file and display the contents of its properties in multiple labels.

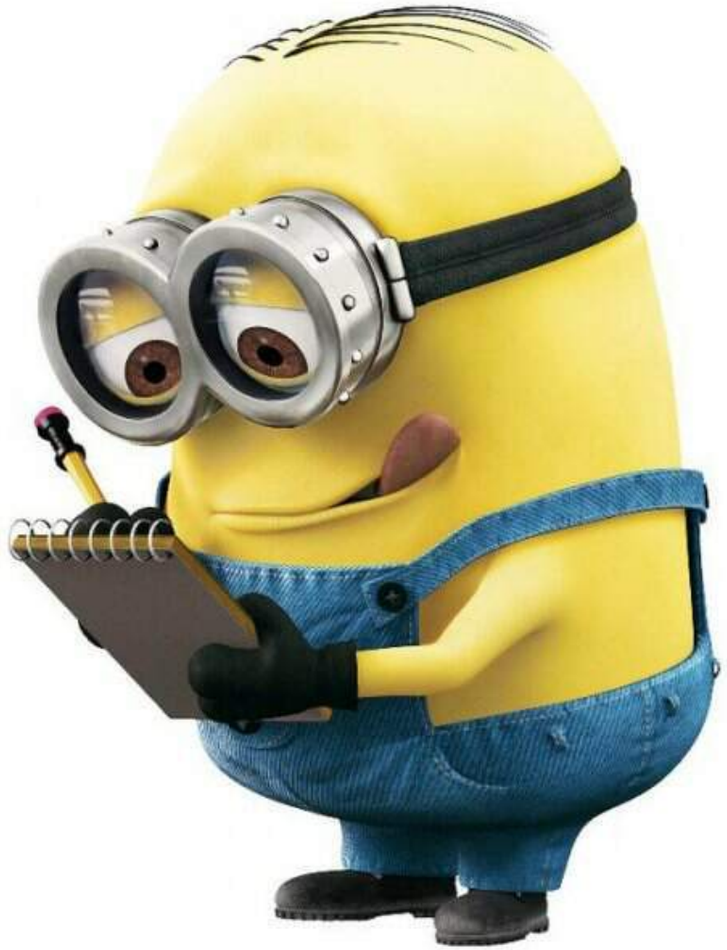
CORE DATA

- Data holding in memory (e.g. array) is volatile
- To save the data permanently a persistent storage like file or database is required
- Core Data is not a database also considering that SQLite database is the default persistent store for Core Data on iOS

CORE DATA – THE STACK

- Managed Object Context, it is a temporary memory area containing objects that interacts with data in persistent store. Its job is to manage objects created and returned using Core Data framework
- Persistent Store Coordinator, SQLite is the default persistent store in iOS
- Managed Object Model, it describes the schema that you use in the app (kinda of database schema)
- Persistent Store, it is the repository that your data is actually stored (a SQLite database or a binary or XML file)

EXPLORING CORE DATA



Q&A

WE HAVE DONE!

THANKS!

GIORGIO NATILI

- Optional Information:
- E-mail: me@webplatform.io
- Source: github.com/giorgionatili/swift (since tomorrow!)
- Twitter: @giorgionatili