Intro to Swift

> Swift is a new programming language for iOS and OS X apps that builds on the best of C and Objective-C, without the constraints of C compatibility. Swift adopts safe programming patterns and adds modern features to make programming easier, more flexible, and more fun.

Apple - About Swift (1)

This section will begin to outline concepts of the Swift programming language including:

1. Swift - The programming language
2. Concepts of programming languages with examples in Swift
3. More Interface Builder - Overview of Cocoa controls and making connections to code

References

(1) About Swift

https://developer.apple.com/library/mac/documentation/Swift/Conceptual/Swift_Programming_Language/index.html

Swift - Overview

https://developer.apple.com/swift/

Swift Reference Book

https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

Style Guide Reference

https://github.com/github/swift-style-guide

Trending Swift Repositories on GitHub

https://github.com/trending?l=swift&since=monthly

Welcome to Swift

Hello and welcome to Swift, Apple's programming language designed to for creating iOS and Mac applications!

> Swift is a new programming language for iOS and OS X apps that builds on the best of C and Objective-C, without the constraints of C compatibility. Swift adopts safe programming patterns and adds modern features to make programming easier, more flexible, and more fun.

from The Swift Reference Book, Apple

[Video] Introduction to Swift

The following is a video from Apple's WWDC 2014 where the introduced Swift.

The Basics

Below, we will list a few basics of the Swift programming language. These examples can be pasted into a Playgrounds window and changed there.

Declaring a variable

Variables represent data within a Swift application:

```
let staticVariable = "This won't ever change"
var dynamicVariable = "This can change"

dynamicVariable = "... you've changed!"
```

Optionals

In Swift, you can define a variable that may be uninitialized (or nil) by using the "?" operator.

```
var newClientName: String?
// newClientName is initialized as nil
```

If you know that an optional variable is currently not nil, you can force unwrapping by using the "!" operator

```
var newClientName: String?
newClientName = "Apple"
let client = newClientName!
```

Performing arithmetic

You can then use variables to perform actions, like math:

```
let a = 1
let b = 2
var c = a + b
print(c) // prints 3

var d = c + a
print(d) // prints 4
```

Control flows

There are a variety of control flows you can use to iterate over data and repeat a particular set of instructions.

These examples will be adpated from the Swift Programming Langugae book [1]

For-in

The For-In loop will repeat its control block until the condition is met.

```
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    println("Hello, \(name)!")
}
// Hello, Anna!
// Hello, Alex!
// Hello, Brian!
// Hello, Jack!
```

For

Swift also supports the more traditional For control loop.

```
// Format
for initialization; condition; increment {
    statements
}
```

Refactoring our last example, we can come up with the same result.

```
let names = ["Anna", "Alex", "Brian", "Jack"]

// initial; continue until;
for var index = 0; index < names.count; ++index {
    println("name at index \(index) is \(names[index])")
}
// name at index 0 is Anna
// name at index 1 is Alex
// name at index 2 is Brian
// name at index 3 is Jack
```

While

Another type of control is the while loop. This loop will continue until the condition is met. Special care should be taken to avoid running an infinite loop.

```
while condition {
    statements
}
```

For example:

```
var counter = 1
while counter <= 5 {
    println("we've counted to \(counter)")
    counter++
}
//we've counted to 1
//we've counted to 2
//we've counted to 3
//we've counted to 4
//we've counted to 5
```

Conditionals

Swift provides a standard set of conditional statements that can be used for checking states of variables and performing optional chains of events.

If Else

The most common conditional statement is evaluating a variable for a particular condition. For example

```
var composerType: String?

if composer == "Bach" {
    composerType = "Baroque"
}
else if composer == "Beethoven" {
    composerType = "Classical"
}
else if composer == "Schumann" {
    composerType = "Romantic"
}
else { // if the above conditions are not met, the execution will come here
    composerType = "Ukknown"
}

print("\(composer) is \(composerType)")
// Beethoven is Classical
```

Switches

> A switch statement considers a value and compares it against several possible matching patterns [1]

The patterns in this case can be a single value, multiple values and ranges of values.

```
switch value to consider {
case value 1:
    respond to value 1
case value 2,
     value 3:
    respond to value 2 or 3
default:
    otherwise, do something else
}
```

As we fill out our example below based on the above format, we are also introducing the range operator **(...)** which is a quick way to include a range of values from x to y.

```
let breakfastCost = 10

switch breakfastCost {
case 0:
    print("Free?!  I'll take it!")
case 1...4:
    print("Sounds good.  I'll buy it!")
case 5...7:
    print("Grumble.  I guess.")
case 8...10:
    print("I'm going elsewhere")
default:
    print("Highway robbery!")
}
```

Storage

Swift provides a variety of storage types (aka **variables**). Let's go over some of them now.

Swift types

We have already used quite a few of these up to this point.

```
let likeCats: Bool = true
let numberOfCatsOwned: Int = 25
let dollarsCatsCostMe: Double = Double(numberOfCatsOwned) * 500.0
let name: String = "Batman"
```

The second to last example **casts** (converts) the numberOfCatsOwned which was an Int to a Double with the syntax Double(Int).

Tuples

Tuples are a group of two or more variables coupled together. They can be different types. Tuples provide a convenient way of passing around a set of information in code.

```
let balloon = (12.0, "Red")

let (diameter, color) = balloon
println("The balloon diameter is \(diameter)")
println("The balloon color is \(color)")

// Output
// The balloon diameter is 12.0
// The balloon color is Red
```

Arrays

> An array stores multiple values of the same type in an ordered list. [1]

Arrays are useful for storing and enumerating collections of similar types.

```
var lbNeighborhoods = ["Downtown", "Bixby Knolls", "North Long Beach", "Wilmore", "California Heights", "Cambodia Town"

for hood in lbNeighborhoods {
    if hood == "North Long Beach" {
        println("\(hood) is home!")
    }
    else {
        println(hood)
    }
}

// Create a new, blank array which will contain Strings
var placesLived = Array<String>()

// Add some data
placesLived.append("Downtown")
placesLived.append("North Long Beach")
```

Dictionaries

Dictionaries are a set of key-value pairs. Each key must be unique.

```
var contacts = ["Jeff" : "201-555-5555", "Joe" : "212-555-6666", "Sarah" : "917-555-7777", "Obama" : "202-555-8888"]

var obamaPhone = contacts["Obama"]
print(obamaPhone)
// 202-555-8888
```

Defining a function

Organize actions into functions:

```
func exampleFunction(thisIsFun:Bool) -> String {

    var returnValue = "No, this isn't fun."
    if thisIsFun {
        returnValue = "Yes, this is fun!"
    }

    return returnValue
}

let enjoyment = exampleFunction(true)
print("Is Swift fun? \(enjoyment)")
```

These are some of the basic tenants of the Swift programming language. As we move the course, you will experience more of the Swift programming language expanding your knowledge and repertoire.

References

[1] Swift Programming Language
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language

Swift Resources https://developer.apple.com/swift/resources/

Style Guide Reference https://github.com/github/swift-style-guide

Trending Swift Repositories on GitHub https://github.com/trending?l=swift&since=monthly

Programming Concepts

Below is a list of concepts associated with Swift and other similar programming languages. The goal of this chapter is to frame your experience and provide a grounding from which to launch.

Having Fun

When you choose what to program, what to build and what to release to the world, realize that it takes time. *A lot of time*. This is all part of the process. Keep at it, try and have fun. The more experience you have the easier it becomes.

Make it easy

As a developer, you'll spend a lot of time reading code. Making code easy to read helps the process of reading and understanding code. Not just for you, but future you. And future employee, too.

```
// This is harder to read
let x = 1 + 2
let y = 3 + x + 2
let z = 1 + 1 + 1 + 1 + 1 + 1 + 6 / 2

// This is easier to read
let x = 3
let y = 5 + x
let z = 9
```

Choose easy. Future you will thank you with a high five.

Commenting

Leaving comments helps future you understand your thought process. As a general rule, keep it short and meaningful. Code should be commented where reading the code may cause confusion.

```
// Example single line comment.  Preferred style.

/*
    Muli-line comment
    Use for commenting out large blocks of code temporarily.
    Don't use long-term.  Makes reverting comments difficult.
*/
```

Importing

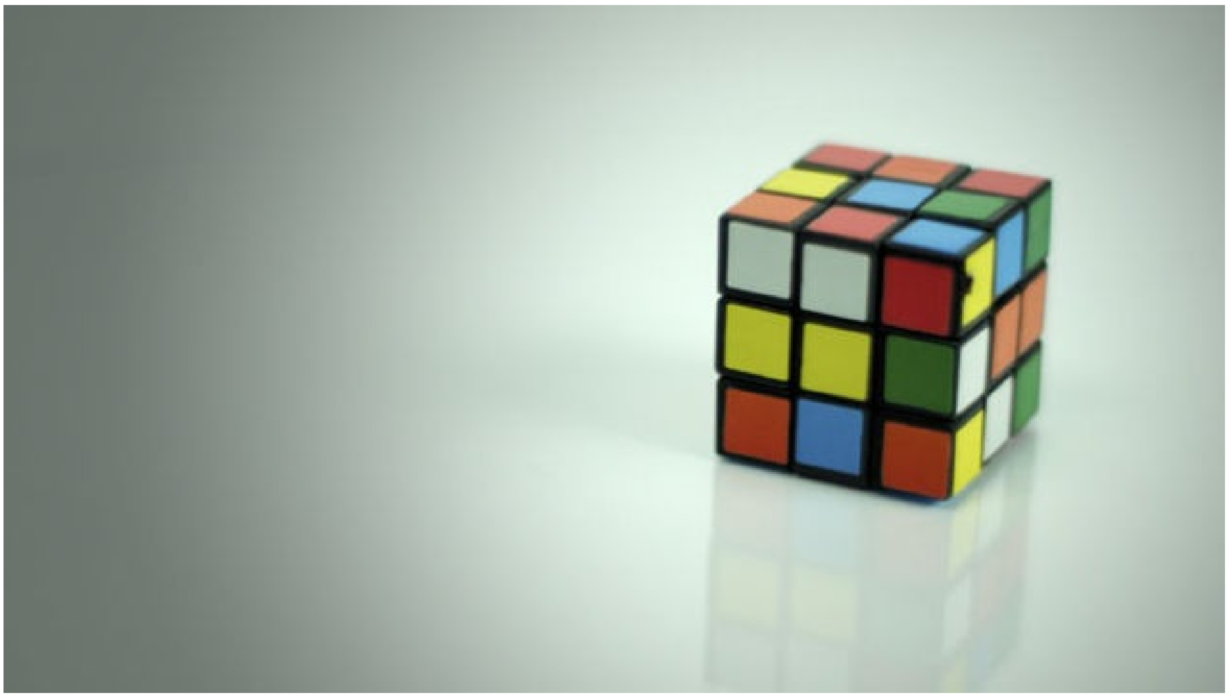
There are numerous libraries that provide added functionality to your code. This is a clean way to import other's code into your own, effectively extending the possiblities of your code with minimal effort.

```
// Example iOS imports
import UIKit // Imports iOS UIKit library for various controls such as UIButton, UILabel, etc.
import Foundation // Imports foundation objects, typically prefaced with "NS" such as NSString, NSData, etc.
```

Break it down

It's easier to program when you understand what you are trying to accomplish. What do you want to happen? Can you break it down further?

A Systematic Approach to Solving Just About Any Problem



Storing Data

Storing data is complicated. Data can be structured in a variety of ways. Data can be sent and received in a variety of ways. Swift provides a variety of buckets you can use for storing data. Read through this code. Copy and paste it into Playgrounds.

```
// Storing one piece of data
let myName = "Jeff"

////
// Storing an array of data
let dangerousSnakes: [String] = ["Rattlesnake", "Cobra", "Python"]

////
// Storing a rectangle
struct Rect {
    var length:Int
    var width:Int
    var color:String
    var area:Int {
        get {
            return length * width
        }
    }
}

let rectangle = Rect(length: 10, width: 10)
let area = rectangle.area // 100

////
// Storing a complex object
class ComplexObject {
    var objectName:String
    var objectType:String
    var objectAge:Int
    var spouse:ComplexObject?

    init(name:String, type:String, age:Int) {
        self.objectName = name
        self.objectType = type
        self.objectAge = age
    }

    func becomeMarried(partner:ComplexObject) -> () {
        self.spouse = partner
    }
}

let me = ComplexObject(name: "Jeff", type: "Human", age: 34)
```

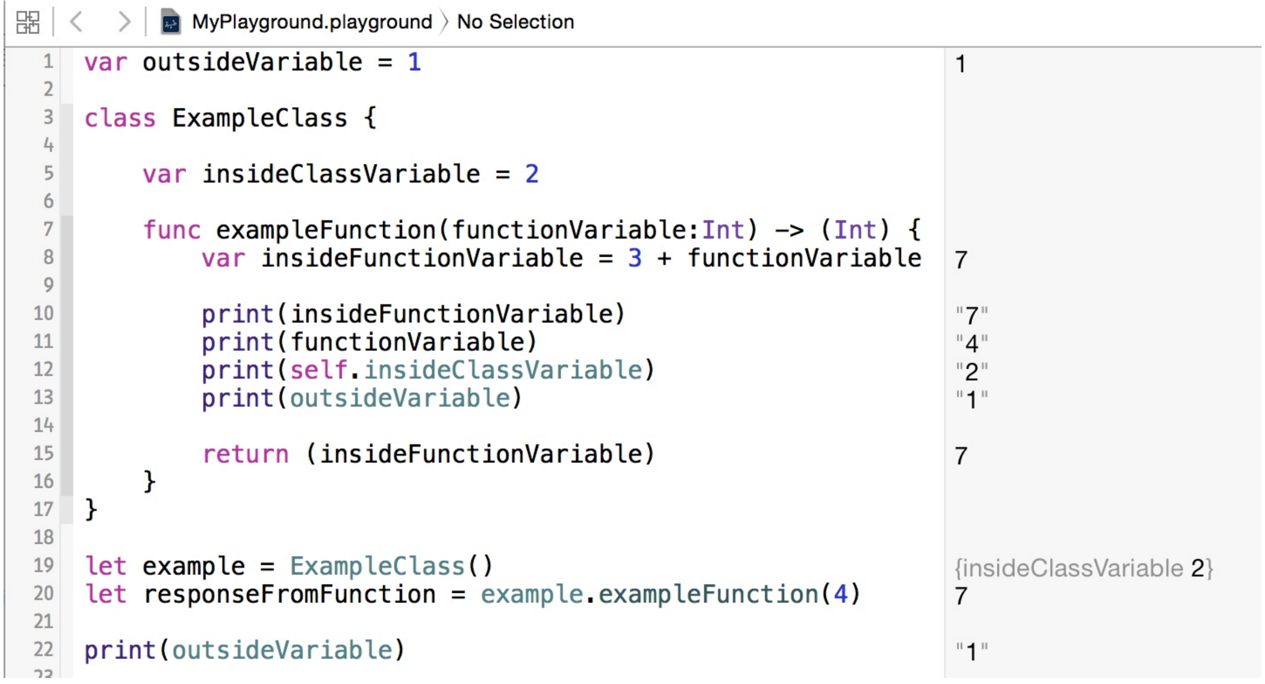
```
me.objectType

let wife = ComplexObject(name: "Katie", type: "Human", age: 32)
me.becomeMarried(wife)
wife.objectType
```

Scope

Scope of a variable refers to where a particular variable's binding is valid. For example, open Playgrounds and paste the following. You can trace the scope in the right-hand panel.

```
var outsideVariable = 1

class ExampleClass {

    var insideClassVariable = 2

    func exampleFunction(functionVariable:Int) -> (Int) {
        var insideFunctionVariable = 3 + functionVariable

        print(insideFunctionVariable)
        print(functionVariable)
        print(self.insideClassVariable)
        print(outsideVariable)

        return (insideFunctionVariable)
    }
}

let example = ExampleClass()
let responseFromFunction = example.exampleFunction(4)

print(outsideVariable)
```

And the output from the right-pane should be similar to this:



```
var outsideVariable = 1                                          1

class ExampleClass {

    var insideClassVariable = 2

    func exampleFunction(functionVariable:Int) -> (Int) {
        var insideFunctionVariable = 3 + functionVariable      7

        print(insideFunctionVariable)                          "7"
        print(functionVariable)                                "4"
        print(self.insideClassVariable)                        "2"
        print(outsideVariable)                                 "1"

        return (insideFunctionVariable)                        7
    }
}

let example = ExampleClass()                                   {insideClassVariable 2}
let responseFromFunction = example.exampleFunction(4)          7

print(outsideVariable)                                         "1"
```

References

Swift Programming Guide - https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

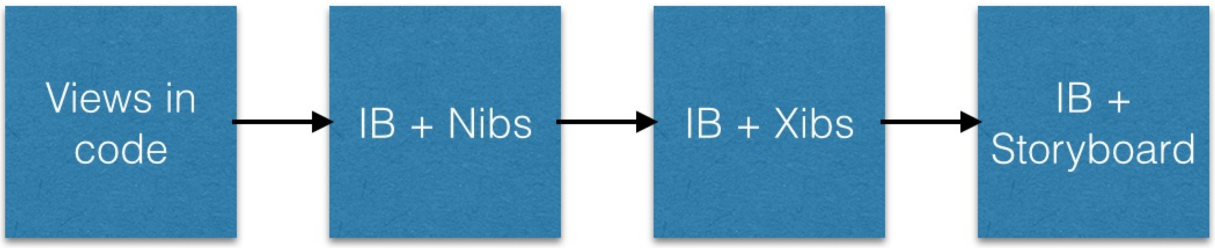
Scope (Programming) - http://en.wikipedia.org/wiki/Scope_(computer_science)

More Interface Builder

The Interface Builder has a variety of tools that allow you to build amazing experience in your app. This section will talk about how to combine IB with source code.

History

Once upon a time, there wasn't an Interface Builder and view programming was done in directly code.

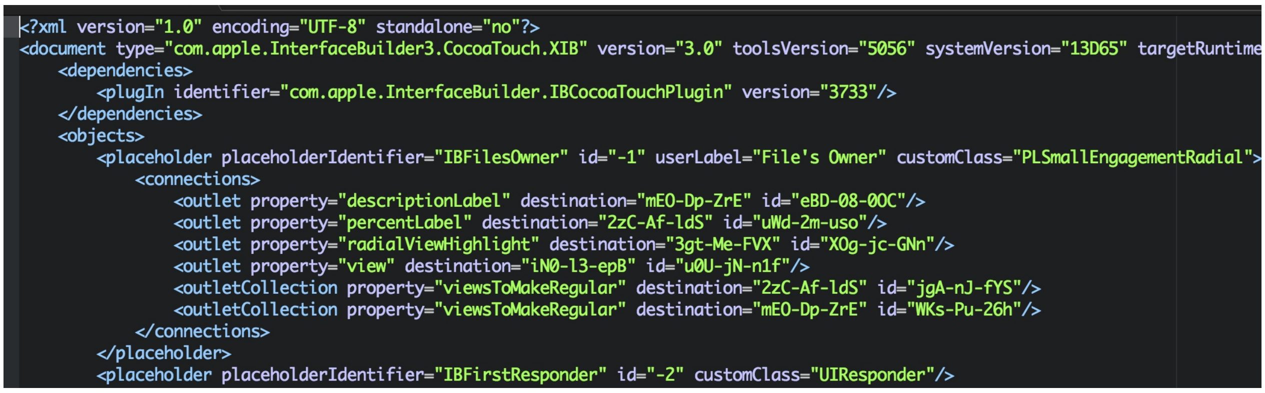


Then along came something called Nibs. Nibs were files that allowed the programmer to accomplish visual editing and were somewhat complicated and not easy to edit outside of Interface Builder (or IB).

XIBs

Even though Apple pushes the use Storyboards, XIBs are still available and are commonly used for reusable views that occur across multiple storyboards or projects.

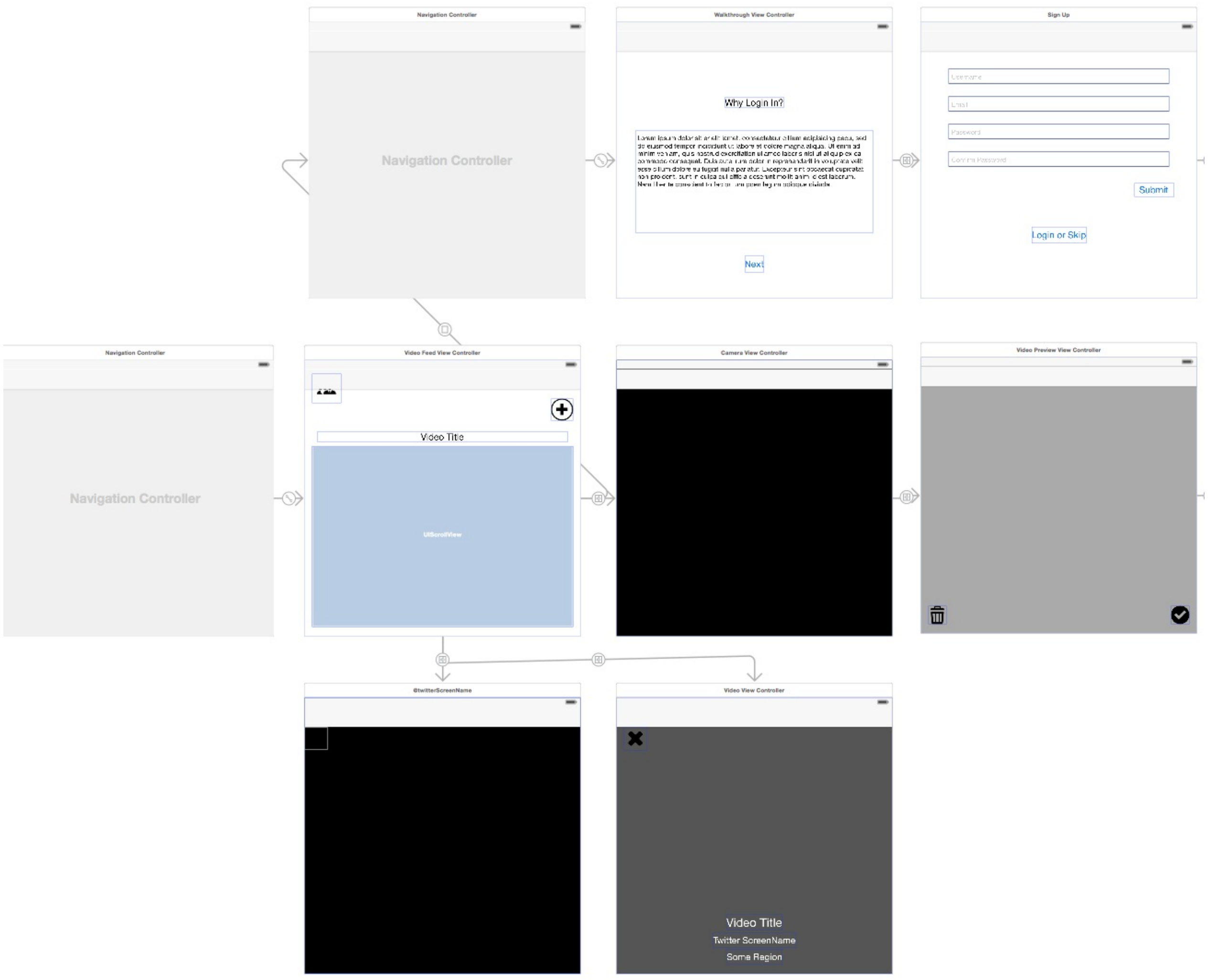
Xibs showed up later. They were more refined and featured XML behind the scenes so you could (if you wanted) view the files with a text editor.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.XIB" version="3.0" toolsVersion="5056" systemVersion="13D65" targetRuntime
    <dependencies>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="3733"/>
    </dependencies>
    <objects>
        <placeholder placeholderIdentifier="IBFilesOwner" id="-1" userLabel="File's Owner" customClass="PLSmallEngagementRadial">
            <connections>
                <outlet property="descriptionLabel" destination="mEO-Dp-ZrE" id="eBD-08-0OC"/>
                <outlet property="percentLabel" destination="2zC-Af-ldS" id="uWd-2m-uso"/>
                <outlet property="radialViewHighlight" destination="3gt-Me-FVX" id="XOg-jc-GNn"/>
                <outlet property="view" destination="iN0-l3-epB" id="u0U-jN-n1f"/>
                <outletCollection property="viewsToMakeRegular" destination="2zC-Af-ldS" id="jgA-nJ-fYS"/>
                <outletCollection property="viewsToMakeRegular" destination="mEO-Dp-ZrE" id="WKs-Pu-26h"/>
            </connections>
        </placeholder>
        <placeholder placeholderIdentifier="IBFirstResponder" id="-2" customClass="UIResponder"/>
```

Storyboards

Last, we have Storyboards - a collection of Xibs within one single file (also XML).



Connections

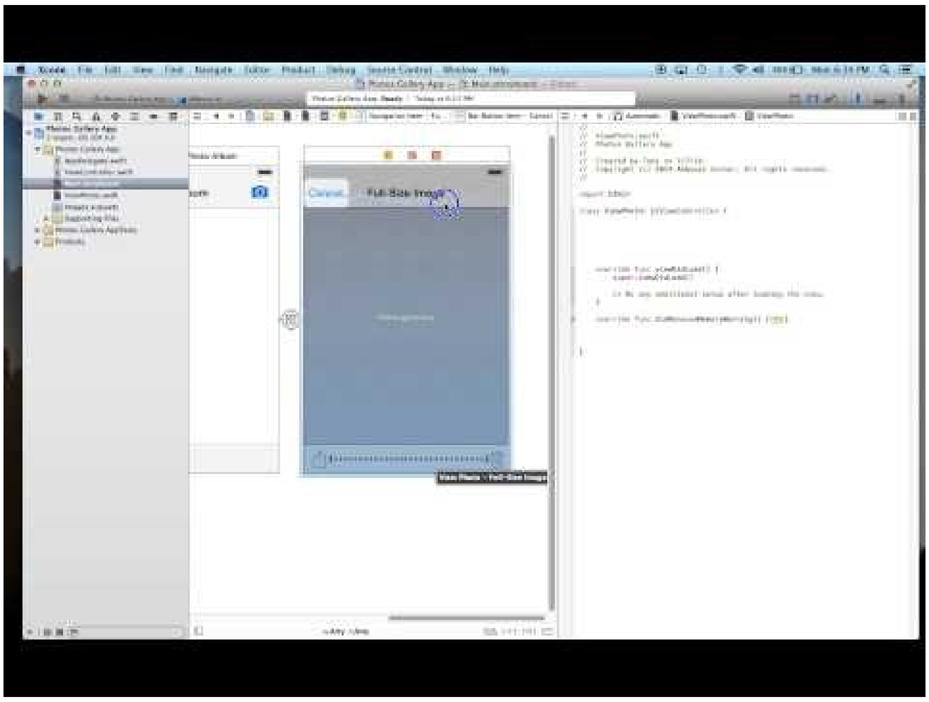
Making connections from the Storyboard into your View Controller code involves dragging outlets from the Utility pane to your control. Make sure you have the parent View Controller selected in the Document Outline.

To begin, you'll need to use your View Controller code and create outlets and actions. Swift uses a special character to denote objects and functions that are available within Interface Builder. That is the @ **symbol.**

```
import UIKit

class YourViewController: UIViewController {
    @IBOutlet var userName:UITextField!
    @IBOutlet var userPassword: UITextField!

    /////

    @IBAction func submitButtonPressed:(sender: AnyObject) {

        // perform some action
        // perhaps with self.userName or self.Password
    }
}
```

Once we have the outlets, we can enter IB, using the Utility pane. We will demonstrate this with a video from YouTube.

[Video] Hooking up outlets and actions with Interface Builder



References

Working with Interface Builder - https://developer.apple.com/xcode/interface-builder/

What's new in Interface Builder - https://developer.apple.com/videos/wwdc/2014/?include=411#411

Deep Dive with Swift

Last week, you started to get comfortable with Swift, concepts that distinguish good developers and were able to work with IB and hook into your code. Well, this week, we're going to go deeper with Swift so buckle up and let's check out:

1. Object Oriented Programming
2. Common iOS Controls
3. Common View Controllers

References

Object Oriented Programming with Swift - http://blog.codeclimate.com/blog/2014/06/19/oo-swift/

Class and Structs - https://developer.apple.com/library/mac/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

Structs, and tuples, and enums, oh my! - https://medium.com/swift-programming/structs-and-tuples-and-enums-oh-my-1b97f82a7339