LECTURE #1

Welcome

First of all. Welcome to Cornell App Development!

The goals for this week's project and tutorial are simple. Xcode will become your best friend. Swift will become your language of choice. Actions will be sent to their targets. Outlets will be updated.

In this tutorial we'll guide you through creating a new project, configuring outlets and actions, and writing the code to connect it all together.

First Xcode Project

In this tutorial, you'll be developing an application that simply displays a message when a button is tapped.

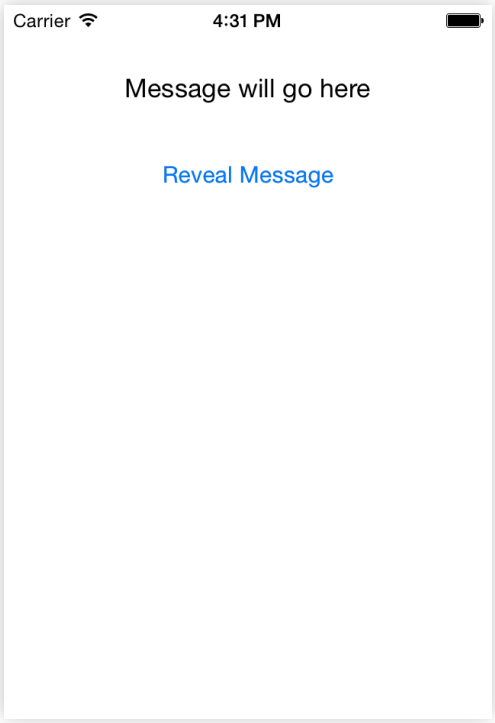

Figure 1-1

To get started, launch Xcode and start a new project. You can either select *"Create a new Xcode project"* from the start window or from the menubar by going to *File>New>Project…*

You'll be prompted to choose a template for your new project. Select *Application* under the *iOS* heading and choose *Single View Application*.

The next screen prompts you for information about the project you're starting. *Product Name* is where you'll name the project, this is generally the name of the app you're trying to create. For *Organization Name*, you can put either your name or the organization you work with. *Organization Identifier* is used to identify your apps amongst every other one in the App Store. This generally follows a reverse DNS format, you can simply put *com.companyname*, in your case this could simply be *com.yourname*. Lastly, select *Swift* as the language, *iPhone* for your device and make sure the *Core Data* checkbox is deselected.

You'll then be asked where to save your project. We recommended you create a separate folder for managing all your projects. Before you save, be sure to deselect the option for creating a Git repository.

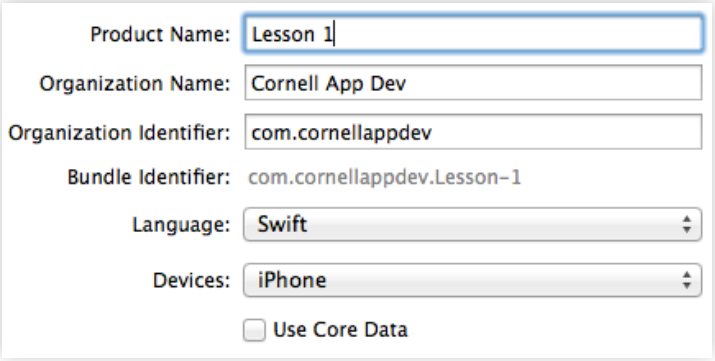

Figure 1-2

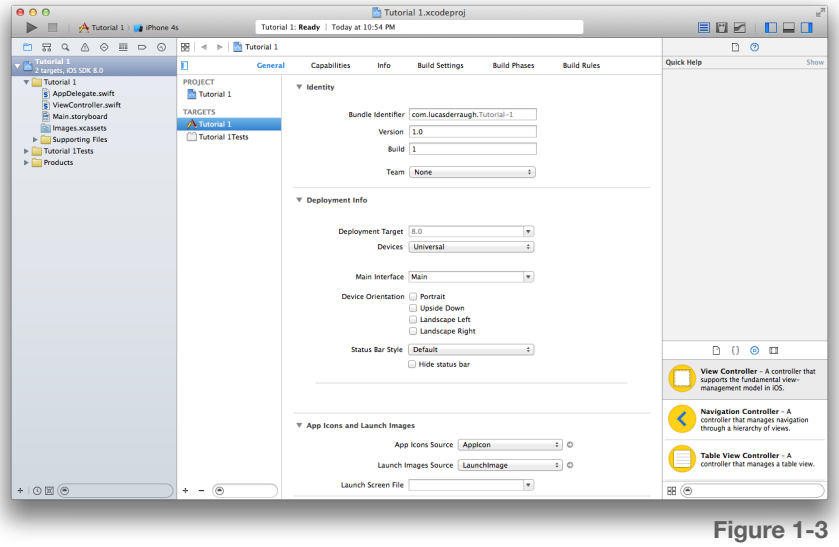

Figure 1-3

Welcome to Xcode

Get used to this window, because you'll be looking at it a lot. Thankfully for you, it's one of the nicest apps Apple probably makes.

On the left hand side, you'll see the *Navigator* pane. This section contains all of your files used in the project. The tab bar in the *Navigator* has a host of other options that we'll talk about later, but feel free to explore.

The right hand side has the *Utilities* pane. This section will be used mostly when designing your graphical user interfaces (GUIs) for your apps. If you want to tweak the color of a button for example, you could do that here. Again, we'll talk more about it when we get there.

The center of the window is where the bulk of the work get's done. This is where you'll alter build settings, write code, and put together what your app will look like.

Over in the *Navigator*, you'll notice a few files to work with. The first is the *AppDelegate.swift* file, any file ending in .swift means that you're working with the Swift programming language in that file. This file deals with scenarios like when your app is first launched, quit, goes into the background/ foreground etc. The next file is *ViewController.swift* which is where you'll write code to control the view (what you see on screen). Feel free to check it out. The last file is *Main.storyboard*, which is where you'll design a lot of how your app looks and is laid out. Let's click on that file, because we want to design what we saw in Figure 1-1.

In *Main.storyboard* you'll be presented with a white screen, that my friends is your app! It's also currently the most boring app ever, so let's change that. Head on over to the bottom right of Xcode and you'll find the *Object Library* (Shown in Figure 1-4). This is where you'll find many of the user interface items to build your applications. In the search field, type in *Label*. Now, drag the label object onto that nice empty white view and position it so that it looks like the label at the top of Figure 1-1. To edit a label, simply double click it. Do the same with the button by searching for a button and dragging it onto the view. Once your interface looks like Figure 1-1, feel free to move on.

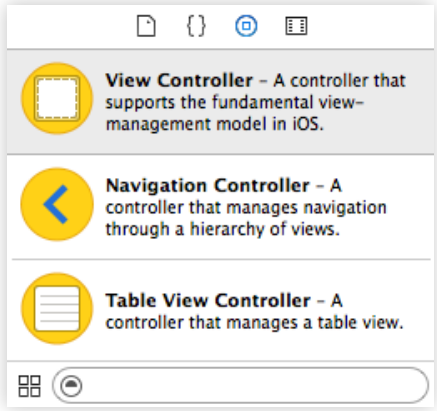

Figure 1-4

To center the label and button, make sure that you dragged them to the center of the view so that the blue guidelines appear on the storyboard and your view. Once you have the label and button centered with the guides, you want to add Auto Layout constraints. In the storyboard, go up to Xcode's menu bar and hit *Editor>Resolve Auto Layout Issues>Reset to Suggested Constraints*, (the option 2nd from the bottom). This will add the appropriate constraints so that the text will be centered in the view. If this doesn't work, feel free to select the *Clear Constraints* option from the menu bar, we'll talk Auto Layout in the future.

Outlet your Actions

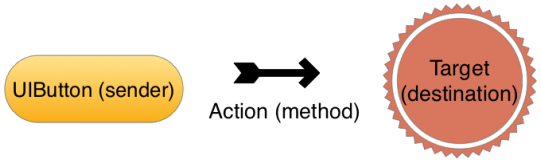

Figure 1-5

Once you have the layout of the application, we need to connect all the objects on the view to the view controller. The view and the view controller communicate with one another through two different mechanism, *Actions* and *Outlets*. *Target-Action* (or *Actions*) is a mechanism commonly used by buttons or text fields to update a target on a specific event. For example, when a button is pressed in iOS, the button will call a method (an action) on it's target to execute some code. The target in this case is usually the view controller. In the case of our Secret Message project we will use *target-action* for our "Reveal Message" button to tell the *ViewController* that it should check the text in the text fields to see if it will unlock the secret message. Before we get into what *Outlets* are, let's make an action (method) of our own.

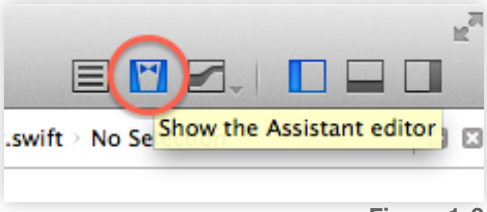

Figure 1-6

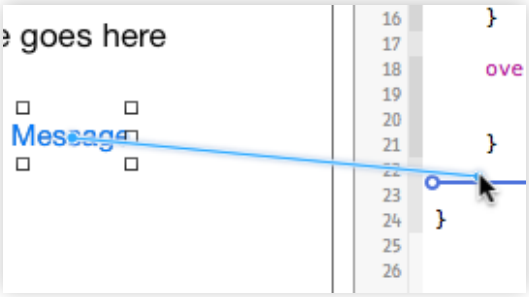

Figure 1-7

While still in *Main.storyboard,* go into Xcode's toolbar in the top right and select the *Assistant Editor* (Figure 1-6,

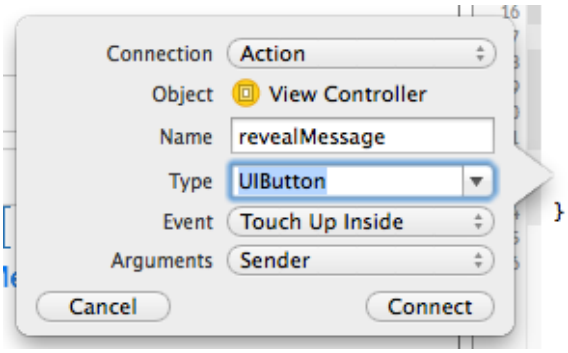

Figure 1-8

⌥⌘↵). You should now see *Main.storyboard* on the left and *ViewController.swift* on the right.

To make a connection between the view controller and your button, simply hold down the *control* key on your Mac and drag from the button into the view controller. When you release the mouse, you'll be prompted to setup the connection. Here you can either create an *Action, Outlet,* or *Outlet Collection*. We want to create a new action, call it something descriptive like "revealMessage" and select the type to be *UIButton*. Once you're done setting that up, hit "Connect" and you've now created a method that will be called anytime the button is tapped.

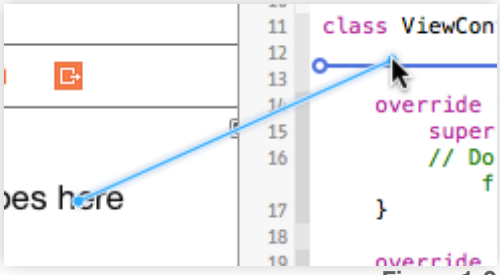

Figure 1-9

Now the last part of this walkthrough is to setup an *Outlet*. An *Outlet* is simply a reference to an object in our view. The view controller needs a way of communicating with things that don't send actions, in our case, we want to know if the text in the text fields are going to match the secret username and password we've setup for the application every time our "Reveal Message" button is tapped. So when the *action* is sent, our view controller would like a way to ask the text fields, "hey, what text do you have right now?" In order for our view controller to do this, it need *outlets* to those text fields.

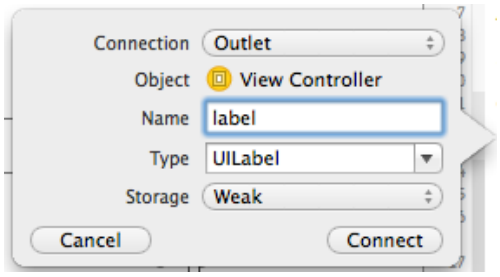

Figure 1-10

Creating an outlet is the exact same process as creating a button, control-drag from the text field into the *ViewController.swift* file (Figure 1-9). Once you let go, configure the new *Outlet*, call it "usernameField" (and "passwordField" for the other), leave the type as UITextField. Lastly, repeat the process for the secret message label along the top of the view.

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet var label: UILabel

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typic
            from a nib.
    }

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

    @IBAction func revealMessage(sender: UIButton) {

    }
}
```
Figure 1-11

Code it up

By now, you're probably asking yourself. Do you code anything to make an app? Of course the answer is, yes! So let's write some. Every time

the user taps the “Reveal Message” button, the *revealMessage* method will be triggered on our ViewController. This is where we will change our *label*’s text. In Swift, it looks like this:

```
@IBAction func revealMessage(sender: UIButton) {
    label.text = "Here's the message!"
}
```

Since we created an outlet to our *label,* we can access the *text* property on the label and set that to a string of our choice. Here we’re setting that to be the string “Here’s the message!”. If you run the application now (hit the big play button in the top left corner), you should find that the app is fully functional and tapping the button will display the new text.