My First App: A Guide to X-Code and Swift for Beginners

Written By: Sabrina Wasserman
Date: Tuesday May 24, 2016

Table of Contents

Getting the Right Hardware

One of the unfortunate parts of programming apps for Apple devices is the hardware requirements. In order to use the X-Code IDE, you must have an apple computer, with an IOS operating system installed.

Installing X-Code

Installing X-Code is a simple process. On your apple device, open up the Mac App Store. Search for "X-Code" in the search bar. Once you find the app, select the install button, circled in red below.

The installation may take several hours depending on your internet connection, so please plan your time accordingly.

While you are installing X-Code, you may be prompted to download X-Code command line tools throughout the download. If this occurs, press "accept", and allow the developer tools to download. This will only happen in a situation where you have an older Mac, or are updating your X-code, so do not worry if this message does not come up.

Once the download is completed, you will see the following screen appear:

Drag the X-Code icon on the left into the applications folder on the right. Now, X-Code will appear in your applications, and you are ready to get started!

Welcome to Swift!

Before starting any project, you will want to get familiar with the programming language. While X-Code does provide simple drag-and-drop features to create user interfaces (more on this later), you should be somewhat familiar with the programming languages you can use to develop IOS Apps.

IOS app development generally occurs in either Swift or Objective-C. In this tutorial, we will be exploring how to use Swift to develop an IOS app. If you would like to learn more about using Objective-C, visit the following page: http://www.tutorialspoint.com/objective_c/.

X-Code has an exciting feature known as a playground, which allows users to create and run code easily, and learn more about the programming language. To create a new playground, open up your X-Code program, and press file > new > playground.

Name MyPlayground

Platform ✓ iOS

Cancel   Previous   Next

Choose a title for your playground, and press "NEXT". Make sure the Platform is set to IOS! Choose a place to save the playground file, and press "CREATE". The following screen will appear:

● ● ●   ⚙ Running MyPlayground

⊞ | < > | 🔷 MyPlayground

```
//: Playground — noun: a place where people can play

import UIKit

var str = "Hello, playground"
```

We have officially created our playground! On the left, you can see X-Code has created some code for you. The grey bar to your right is where any code you run will appear.

Now we can learn about some of the basics of swift. Similarly to python, Swift does not require the user to specify the type of variable when it is created. The user must, however, identify that what they are declaring is a variable by using the keyword "var".

To declare a variable, the structure is as follows:

var example = "Hello World!"

I have used "Hello World!" as an example, but you could set that variable to equal anything. You can also replace the world example with whatever you would like your variable to be called, so long as you keep the "var" keyword at the beginning. You can replace the "Hello World!" text with anything you want - an integer, decimal number, or anything. Just remember - regardless of what you chose as the type of your variable, you must keep that type consistent throughout the program. For example, I could not set my above example variable to equal 5, since initializing it as "Hello World!" told the console I wanted it to be a String, or word, value. Try creating some variables in your console!

It would take to long to cover the basics of Swift in just one document - there is so much to learn! Loops, methods, and object-printed programming commands are a lot of information to fit into one small info- book! To learn more of the basic swift commands, you can visit the following link: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html.

My First App - Happy Halloween!
For the purposes of this tutorial, we will be making a silly Halloween app. In the app, we can upload pictures of ghosts, and have a "speak" button where the ghost will be able to make noises, and say things to us, like "boo!".

While the app may not be the most useful, it's creation will show us some of the key aspects of building an app. We will develop the user interface, create buttons with actions, add pop-up notifications, and even include sounds in the app!

To the left, you can see a quick preview of what our app will look like.
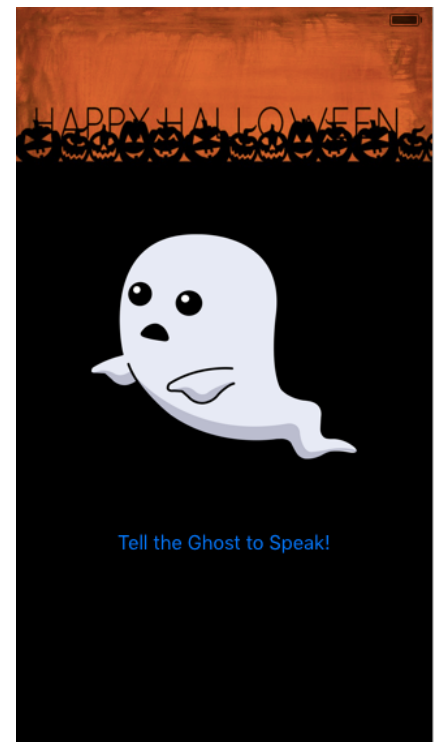
Creating our App Project!

Now that you've learnt the basics of Swift and X-Code, it's time to actually create our app! In X-Code, files that save app codes are known as projects. Creating a new project is very simple. You can simply click on the "create new project" button on the X-Code welcome screen, or press File > New > Project.
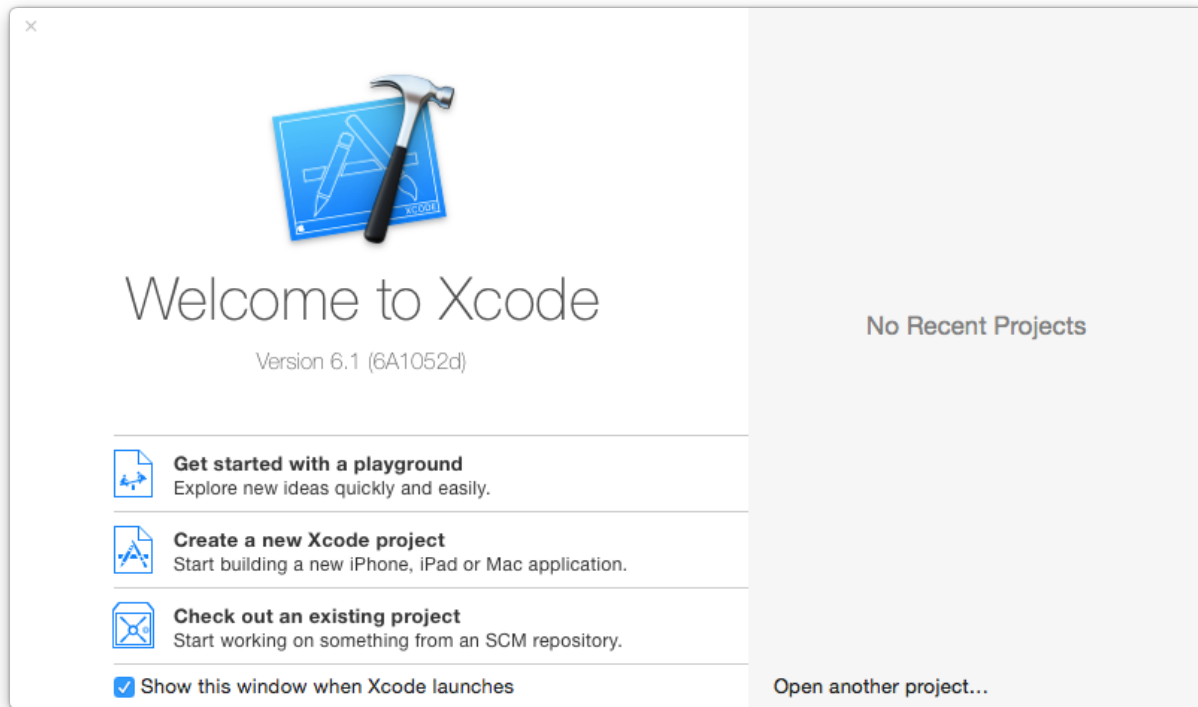
X-Code will now ask you what type of application you would like to make. For our purposes, we will be creating a Single-View Application. This just means that our app will have one page. Select "Single View Application" from the menu box, as shown in the diagram to the right.
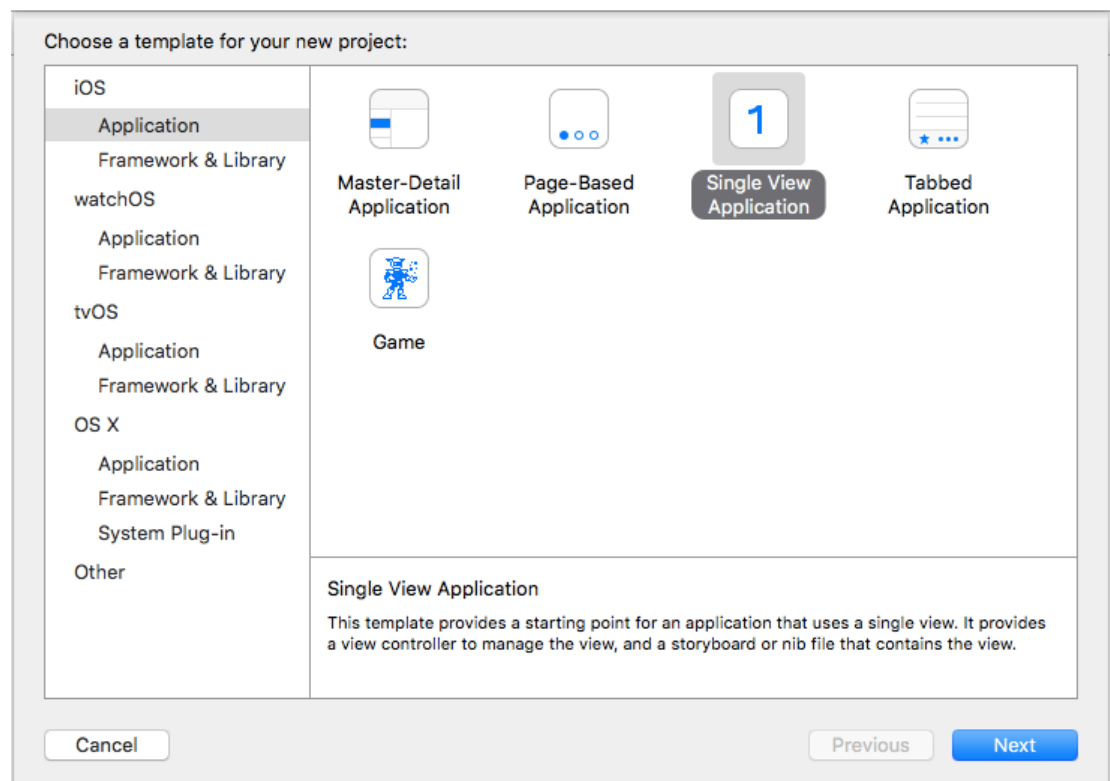
Press "NEXT". Now, you can create a title for your project. In this tutorial, we will be making a silly Halloween app, so you may wish to title this app something that encompasses Halloween. You will see spaces for "Organization Name" and "Organization Identifier". Unless you intend to publish this app, these spaces are generally irrelevant. Simply fill in an organization name you like (choose something awesome!), and the identifier should fill itself out automatically.

Make sure that the language you have selected is Swift, and that the "Devices" section is selected to iPhone! You will see three checkboxes on the bottom - you can check all three of them. Press next, and choose a place to save your project.

Congratulations! You have just created your app project!

Intro to X-Code

Now that you have created your X-Code project, you likely noticed that it looked far more complicated than it did when we were simply using a playground. But not to worry! Let's take a quick tour of X-Code.

To following diagram gives you a brief tour of the X-Code layout:

Toolbar   Editor area

Navigator area   Debug area   Utilities area
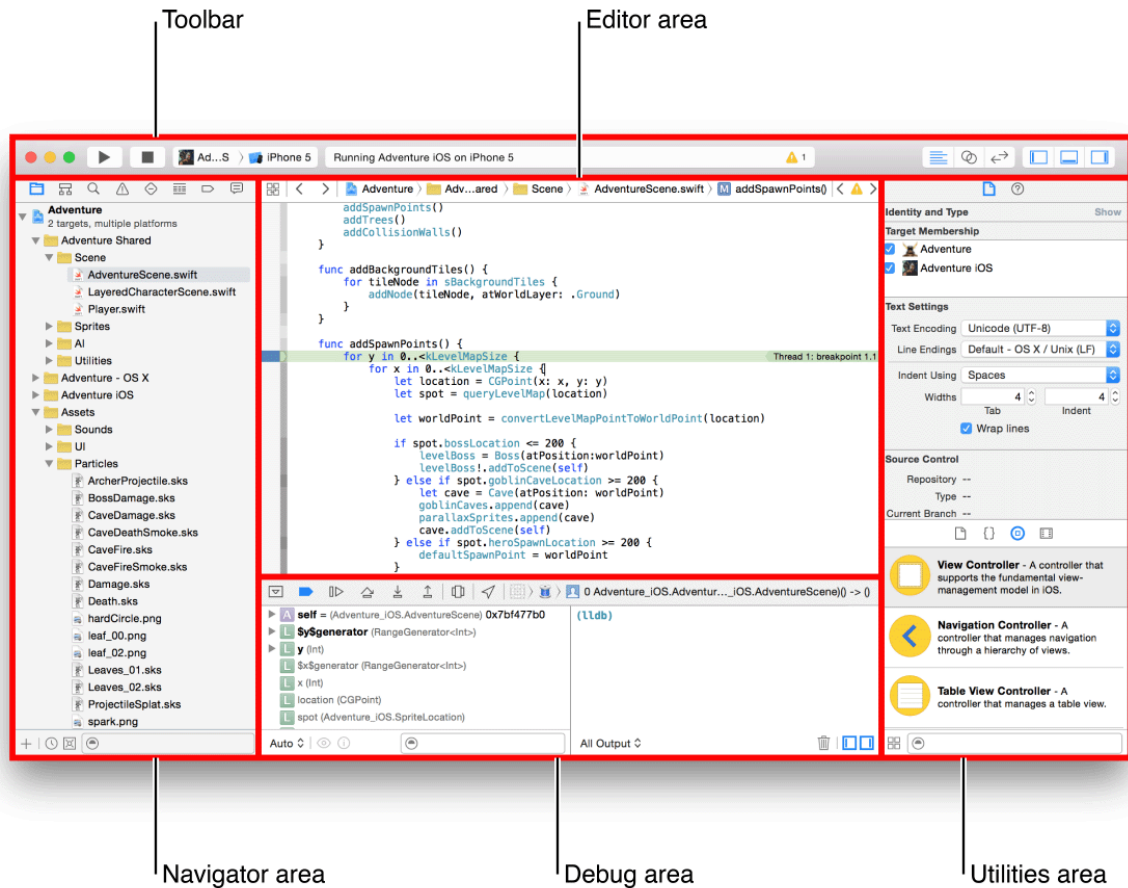
Here's a quick guide on what each of the panes do:

Navigator Area: shows you all the files in your X-Code project. Any pictures or code files you have in your project will be shown here. You can also create new files or drag new ones into your project via this pane, but we will get into that a little bit later.

Debug Area: The debug area shows us any information we have printed to the console from our code. It will also show us any errors we may come across throughout the project. This section will quickly become your enemy once errors start to happen while we program 😡.

Utilities Area: This area will be very useful once we start to develop our app in more depth. It will provide us with detailed descriptions of the object we are making, and lets us connect our user interface (UI) to our code. The utilities area also allows us to create objects for our app, like navigation bars, pictures, or check boxes!

Editor Area: The editor area shows us the file that we have highlighted, or selected, from the navigator area. It can show us code, as well as our app's user interface.

Toolbar: You've probably noticed that the toolbar contains a play and stop button. This lets us run our code, and opens an iPhone simulator to run our program on! The toolbar also lets us customize the device we are designing the program for (iPhone 5, iPhone 6s +, iPad 2, etc.). It also allows us to change out pane views, but we will get into that later.

To get a more in-depth tour of X-Code, visit the following video: https://www.youtube.com/watchannotation_id=annotation_2506386695&feature=iv&index=3&list=PLMRqhzcHGw1ZF7Vd Tt2EALt_5i6RELc0k&src_vid=G44oIktEvg8&v=CYZYcEi8Ll4

First Steps of UI: Fixing the Layout

It's time to start our user interface, or UI. User interface in a nutshell is what the app users will see. It's what the app will look like when it is actually running on an IOS device. Since X-Code very easily allows us to connect our user interface with our code, creating the user interface first lets us visualize the project better, and helps us understand what we are programming our interface to do.

In your navigation area, you should see a file titled "Main.storyboard". This is your app's storyboard file, and basically shows us what the user interface of the app looks like. Right now, your user interface probably looks something like this:
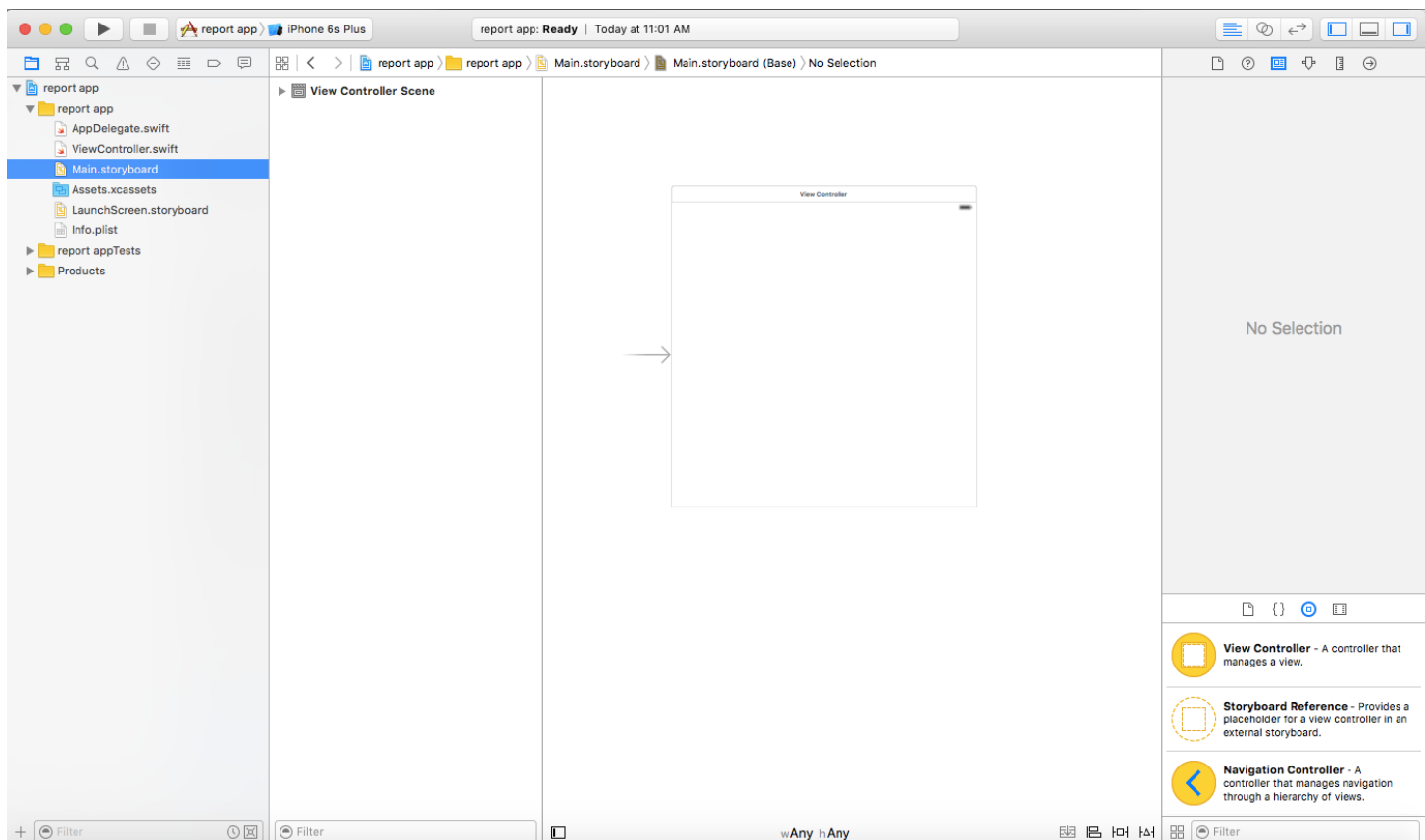
See that rectangular box labeled view controller? That is our app's only page. Right now it is empty, since we have not added anything into it yet.
Take a look to the left of the view controller. Do you see the grey arrow that appears to be pointing to the box? DO NOT TOUCH IT! This arrow tells the console which page to load first when X-Code decides to launch our app. Since our app only has one page, it should only be pointing to this box. If you delete this box, the console will not know where to start launching the app, and cause errors that will make your app fail to build. In short, DO NOT TOUCH IT!

You might've also noticed that the shape of the app's page doesn't exactly emulate that of an iPhone app. This ambiguous shape is used by programmers when they want to create their app for multiple platforms - iPad, iPhone, and even the Apple Watch. Since our app is just for the iPhone, we can change the view controller's layout to be more suitable for the iPhone.

Select the view controller by clicking on its rectangle. You will notice that the top of the Utilities area will change according to the object we have selected. At the top of the utilities pane you will notice a list of tabs. Select the first one, that looks like a piece of paper. This tab is highlighted in the image below.

Scroll down to where the utilities pane says "Interface Builder Document". Find the checkbox labelled "Use Size Classes" and deselect it. A Pop-up notification will ask you if you are sure about this decision - just press disable and you will watch as the view controller now shifts to the size of an iPhone screen.
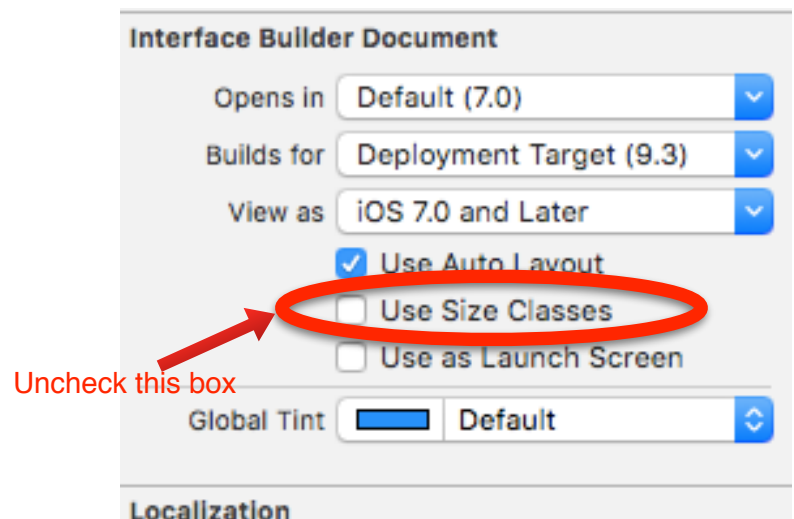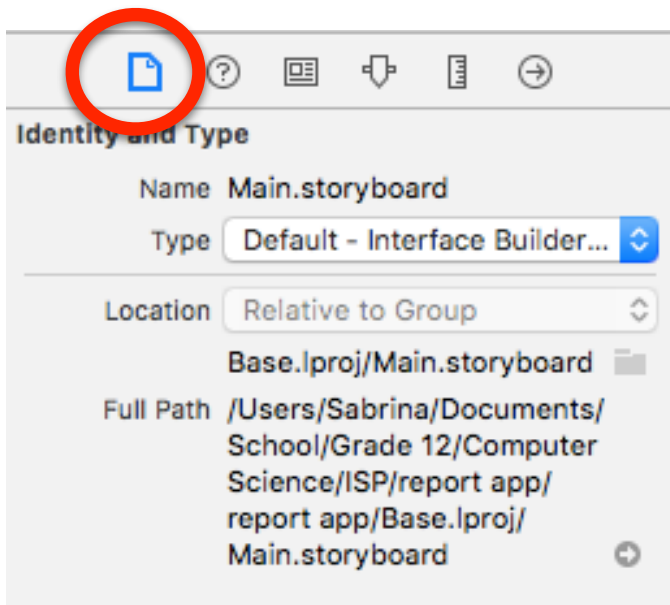

Now that our view controller is done, we can start to add features to our app!

Adding Images

Now that our view controller's layout is set the way we want it to be, we can start to add items to our app. Let's start with the simplest thing - images! Take a look at your Utilities Pane. Towards the bottom, you will see a small menu, shown to the right. Select the third tab on the menu toward the top, circled in red.

Now you will see a multitude of objects that X-Code has pre-installed for us to use. Using these objects, we can add images, extra pages to our app, buttons, and much more without any programming required.
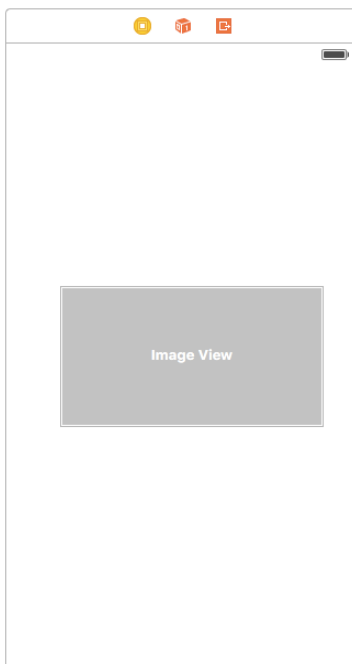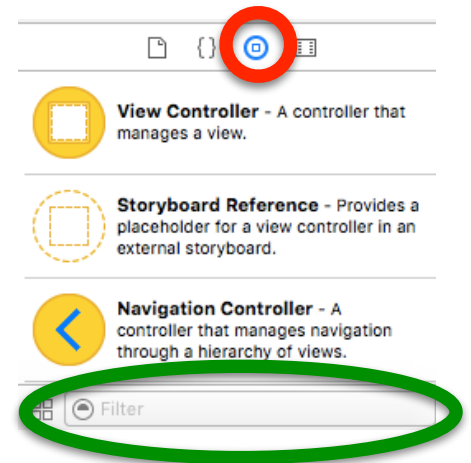

Go the the search bar at the bottom of this menu, circled in green. Type the keyword "image" into the search bar. You will now see only "image view" left in the search menu. Press and hold that menu option, and drag it into your view controller box. It should look something like the view controller screen on the left now.

Here comes the fun part - picking images! Whether you want to draw your own picture or get it from the internet is your choice. Since this is a Halloween app, let's start with something simple - a picture that says Happy Halloween. I found the image I used in the above example from the following link: https://www.improveit360.com/happy-halloween-from-improveit-360-2/.

Whichever image you choose, save it to your computer. Then, drag it into the folder for your app within your navigation pane. You should see the file appear within the folder in the navigation pane. Now the image is saved into our project, and we can easily add it into our project.

Now that your picture is saved into the project, we can assign it to the image view you just dragged into your view controller. Select the Image View box in your view controller, and head over to your Utilities pane. In the pane, select the attributes inspector (the one that looks like a little shield), which is circled in red below.
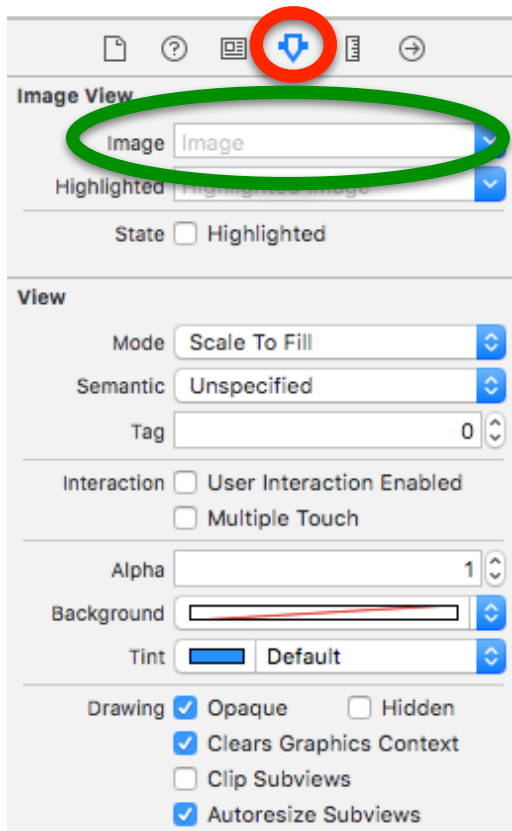
Make sure that your image view is selected, or these menu options will not be the same! Take a look at the section labelled "Image", which is highlighted in green.

Since X-Code already has our image saved into the project, we can simply press the drop down menu button to assign our image to this image view. Press the blue arrow on the side of this box, and the image you dragged into your navigation pane should appear! Select this image, and now your image view should have your "Happy Halloween" picture in place of the grey box.

You will likely have to re-adjust the size of the picture for it to look as you would like. Simply drag the edges of the image to achieve this re-sizing.

If you have done this correctly, you should have your image in the app! Your app probably looks something like the image on the bottom left. Drag the Happy Halloween picture up to the top, so it can act as a pretty header for your app!
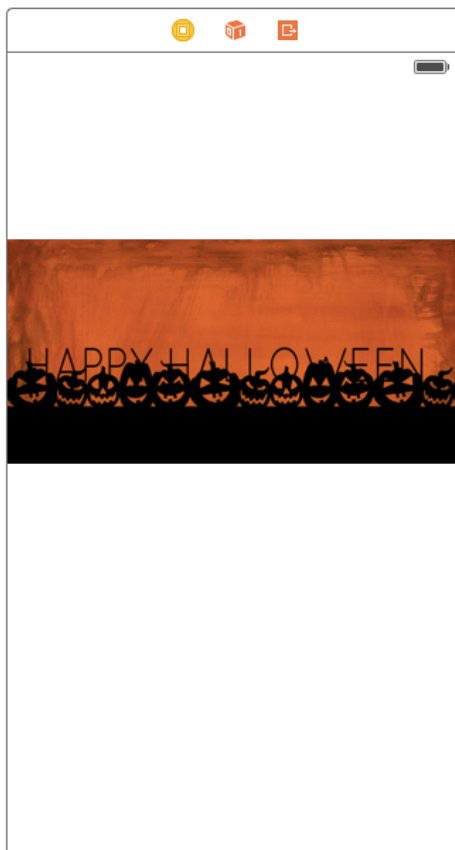
Now that you know how to add images like a pro, repeat the same process with a picture of a ghost! The ghost image I used in my above example can be accessed from this link: http://weknowyourdreamz.com/image.php?pic=/images/ghost/ghost-06.jpg. Feel free to find your own image of a ghost, or to draw your own and import it in!

Creating the Button UI

You've probably realized at this point that we have not yet programmed anything for our app yet - and that's ok! We are about to get to the programming section.

In this app, we will create a button that allows the ghost to "talk to us". In reality, it will launch a pop-up notification that will say "BOO!". We will start by creating the UI of the button, similarly to how we did with the picture, but we need to do some programming after to tell the console what to do when the button is pressed.

Go down to the utilities pane, in the same area where we searched for the image view. In the search bar, type in the keyword "button". A few options will appear. Select the first option, that says "Button". It is circled in red below.

12

Just like before, drag the button into your view controller, under the image of the ghost you added. You'll notice that unlike the image view, the button is simply a line of blue text that says "Button".

Let's start by changing what the actual button says. Double click on the button, and you will see that you can now change what the text of the button says, as shown in this diagram:
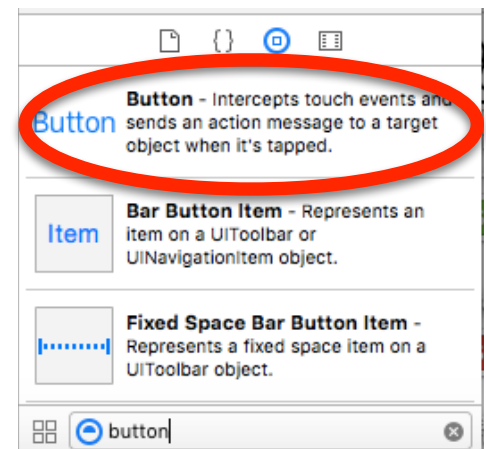
Change the button text to say "Speak to the ghost!". That way the user knows that when they press the button, they will be able to speak with the ghost!

Right now our button is blue, but we can change that to be more Halloween like. Let's make the button's text orange! Select the button, and go to the utilities pane to the attributes inspector, the same inspector we went to when assigning images to our image view. Here, you will see an option that says "Text Colour". Change the colour to whatever you desire! (I recommend orange since it's a Halloween app, but this is your app!).

Now that we have our button in our app, it's time to program it!

How to Run Your Project in the Simulator
Now that we have added images and buttons to our app, we can run it in an iPhone simulator to see how it will run. In the toolbar of X-Code, press the big play button on the top right corner. This will launch the simulator, and allow you to run your code! When you are done running the code, press the stop button in the toolbar located right next to the play button.

Connecting the UI to the Code
If you take a look at your navigation pane, you should see a file labelled "viewcontroller.swift". This file was created for us when we made our app, and is already connected to our view controller. The one problem - it's not connected to any of the objects we made in our code! That means right now, our code file cannot take action commands from our button. But we can easily change this!

Take a look at your toolbar in X-Code. On the top right, you will see a series of buttons with arrows, circles, and lines, which is circled in red below:
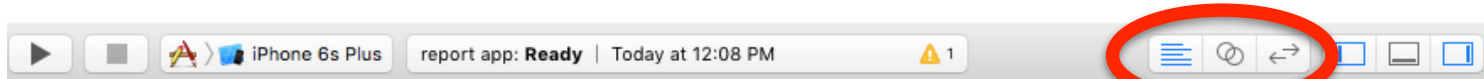
Select your view controller, and press the circle button in the middle of the 3 circles above. You will see a code file appear on the right of the storyboard.

Go back to your storyboard. Press and hold the control button on your keyboard while selecting the button we just created. Keep your finger on the control button. If you start to move you mouse, you will see a blue line appear. Drag that line all the way into a blank space in your code file, as shown below.
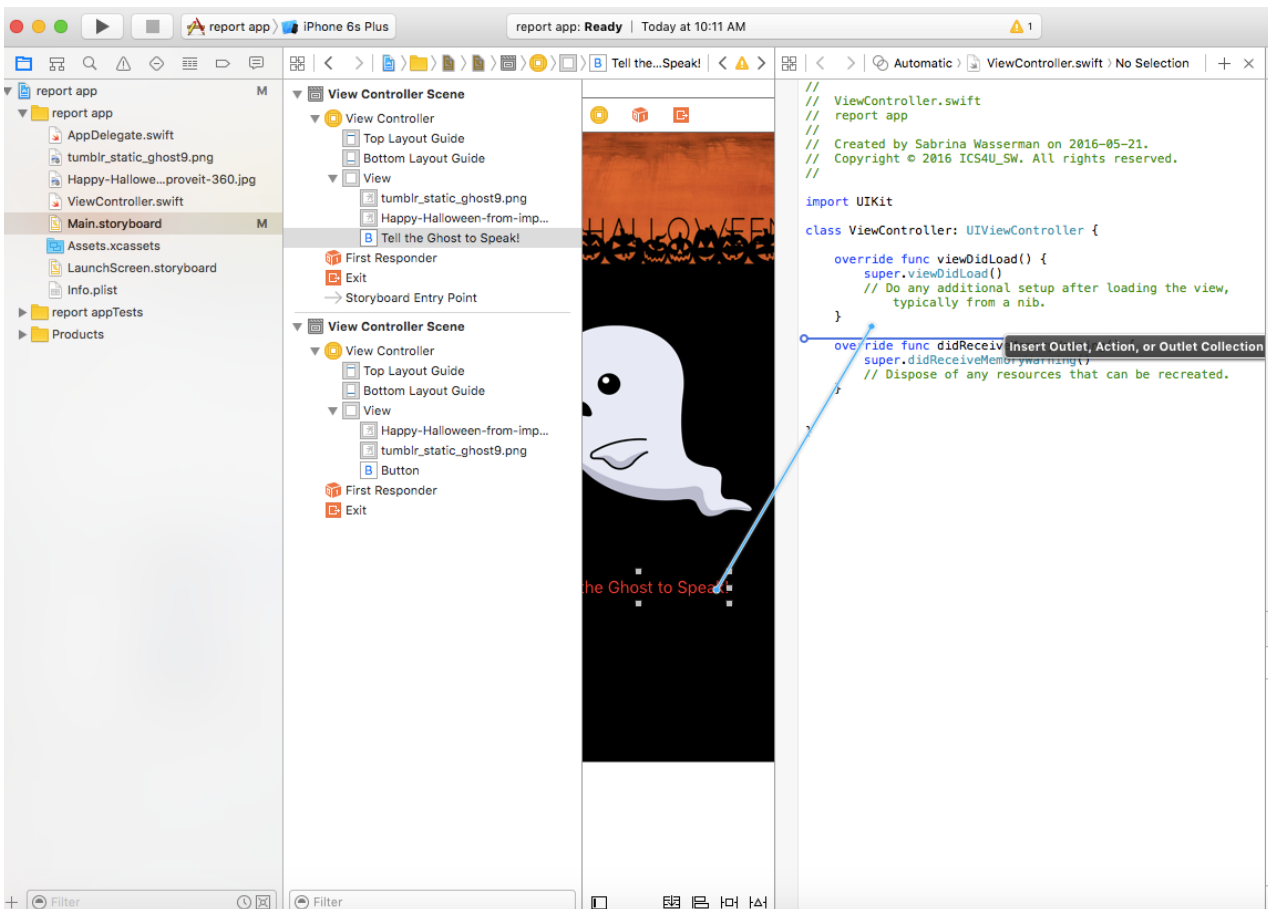
Release your mouse and the control button once the line is in this position. You will now see the pop-up shown to the right appear. Change the connection to "Action", and set the title to speakButtonPressed. Press "Connect" Once you have changed these settings. You will notice that a new method in the code has appeared. This method will control what happens when the user presses that button - so now we can program the notification!
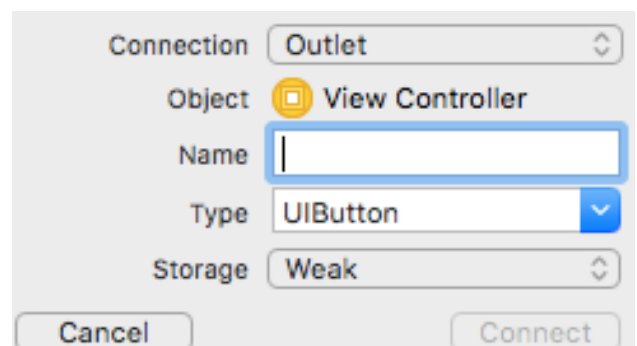
Creating the Pop-Up Notification

In your navigation pane, select the file titled ViewController.swift. You should now see a code file where your storyboard was before. Inside the curly brackets of the new method we created above, insert the following code:

```swift
@IBAction func speakButtonPressed(sender: AnyObject) {
    let alert = UIAlertView()
    alert.title = "The Ghost Says..."
    alert.message = "BOO!"
    alert.addButtonWithTitle("close")
    alert.show()

}
```

This code is what allows us to create the text alert, or pop-up notification. The header of the notification will say "The Ghost says…", and the message will say "BOO!", as you can see in the code above. Now that we have all of this programmed, let's try running the code. Press the button, and this should appear!

15
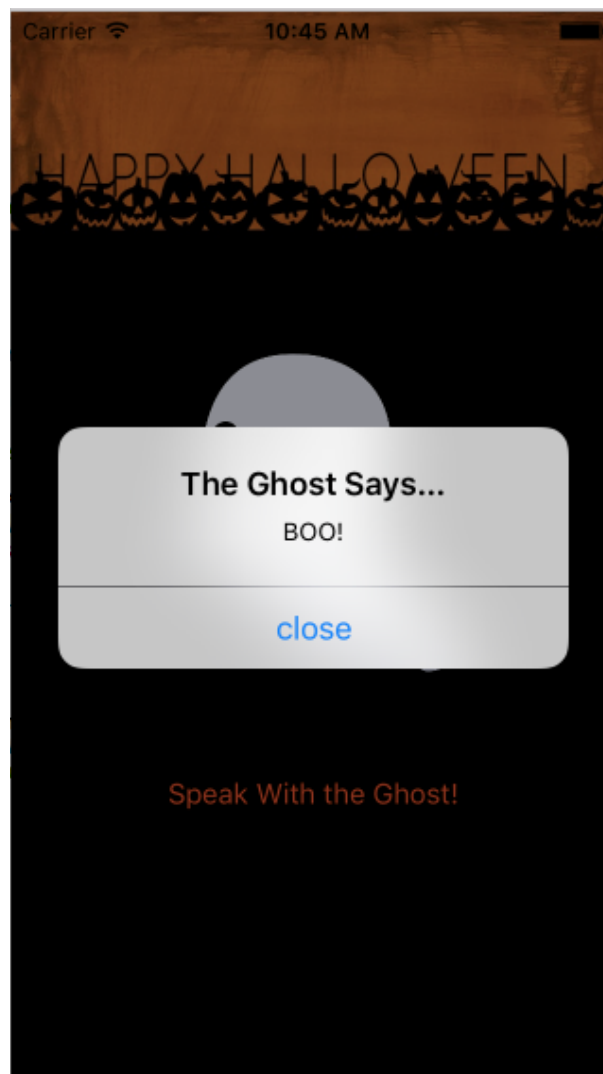
If you do not see this, you may be experiencing some errors, which we will get into in a moment.

Common Errors in X-Code

With all programming situations comes the potential for errors. Here are the most common error you are likely to come across when creating your simple app, and how to fix them.

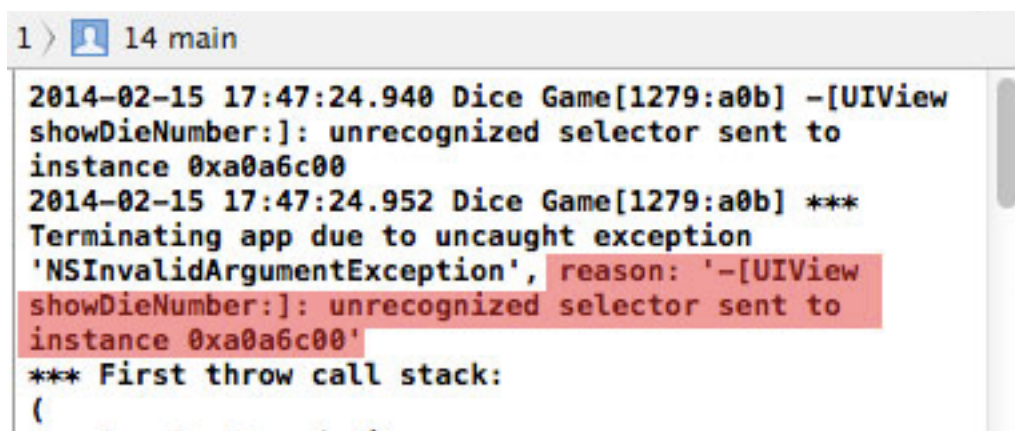**Error One: SpringBoard Failed to Launch Application With Error: 0**
This error will occur when you try to run your application. To fix, simply go into your IOS Simulator, and press the button that says "Reset Content and Settings".

Error Two: Thread and Breakpoint Errors
Breakpoint errors can be very frustrating, but are simple to deal with. They generally occur if you accidentally set a breakpoint at some point within your code file, which causes the simulator to stop when that point in the code is reached. To fix this, go to the "Debug" menu at the top of your X-code window, and press "Deactivate Breakpoints". This should fix the error.

Error Three: Unrecognized Selector Error
This error occurs when an object has been created, but doesn't have a corresponding method. The great thing about this error is than it will show the name of the object without the method, so you can easily fix it. Pictured below is an example of this error where the object was called "ShowDieNumber". To fix this error, find the object that the console has specified and add a method for it.


```
1 ⟩ 👤 14 main

2014-02-15 17:47:24.940 Dice Game[1279:a0b] -[UIView
showDieNumber:]: unrecognized selector sent to
instance 0xa0a6c00
2014-02-15 17:47:24.952 Dice Game[1279:a0b] ***
Terminating app due to uncaught exception
'NSInvalidArgumentException', reason: '-[UIView
showDieNumber:]: unrecognized selector sent to
instance 0xa0a6c00'
*** First throw call stack:
(
```

Error Four: Missing Braces
If you are missing a brace to close off one of your methods, X-Code will give you error warnings for all of the methods below that one, as pictured below. To fix, simply check to make sure all of your brackets are in place correctly to remove any errors.
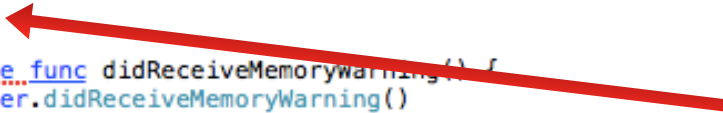
```swift
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
    @IBAction func speakButtonPressed(sender: AnyObject) {
        let alert = UIAlertView()    ⚠ 'UIAlertView' was deprecated in iOS 9.0: UIAlertView is deprecated.
        alert.title = "The Ghost Says..."
        alert.message = "BOO!"
        alert.addButtonWithTitle("close")
        alert.show()


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

}
```

Missing brace to close off this method leads to errors in the next method

Reference Links

1. http://codewithchris.com/xcode-errors/

2. https://www.safaribooksonline.com/library/view/learning-the-ios/9781449399078/apb.html

3. http://stackoverflow.com/questions/19700382/how-to-display-ios-push-notification-popup

4. https://developer.apple.com/library/ios/documentation/Swift/Conceptual/
   Swift_Programming_Language/Functions.html

5. https://www.hackingwithswift.com/read/0/1/how-to-install-xcode-and-create-a-playground

6. https://developer.apple.com/library/ios/recipes/xcode_help-image_catalog-1.0/chapters/
   AddingImageSets.html

7. https://developer.apple.com/library/ios/recipes/xcode_help-image_catalog-1.0/chapters/
   AddingLaunchImagestoanAssetCatalog.html