# Introduction to Xcode and Swift

This workbook provides a basic introduction to the Xcode IDE and the Swift programming language. It includes explanations, code descriptions and code examples, which you should read. It also includes activities, for example to create or modify some code. The activities reinforce what you have read, but also give you vital coding practice. The workbook is design to be read in sequence, if you skip parts it may make it harder to understand later parts.

Whilst the workbook includes everything you need to cover, you may find it useful to refer to other sources of information. There is a lot of information and documentation available at <https://swift.org/documentation/> including a guide <https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html>

The codeacademy website has a course on Swift, but it is new and doesn’t contain many lessons at the moment. Our library also has books on Swift.

There are different versions of Xcode and Swift and we may not have the latest version in the labs, so watch out for incompatibilities!

## Xcode

Xcode is an Integrated Development Environment for developing macOS applications. It supports the development of applications for macOS, iOS, watchOS, and tvOS. Xcode supports a variety of programming languages, including C++, Java, Python and Swift. It also includes an interface builder and a device simulator.

## Swift

Swift is a high-level programming language which can be used for developing applications across a range of Apple devices, including macOS, iOS, watchOS, and tvOS.

Swift is supposed to make it easier for non-programmers to develop applications. The code structure was developed specifically for Xcode/Apple programming environment.

Swift development includes a Playground which acts as a simulator to see test code.

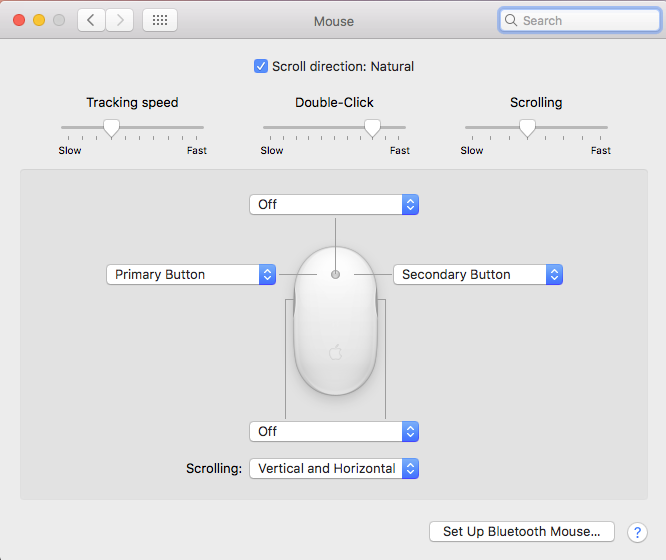
## Getting Ready

First, we need to check a couple of things…

1. Check to see if you have access to your student folder. If not, you will need to do your work on the Mac (create a new folder on the desktop) and then copy it to a pen drive / cloud drive at the end of class. Remember to delete the folder from the desktop.
2. Set the right mouse button as ‘secondary’, if it is not already set. To do this, click on settings:



Select mouse, then check / set the right mouse button:



1. If you cannot see the Xcode icon on your desktop, click on the Finder icon



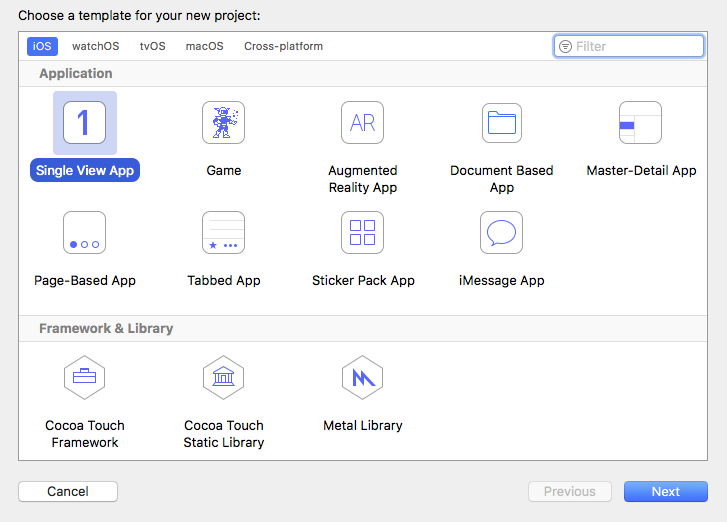
then select ‘Applications’ and look for the Xcode icon



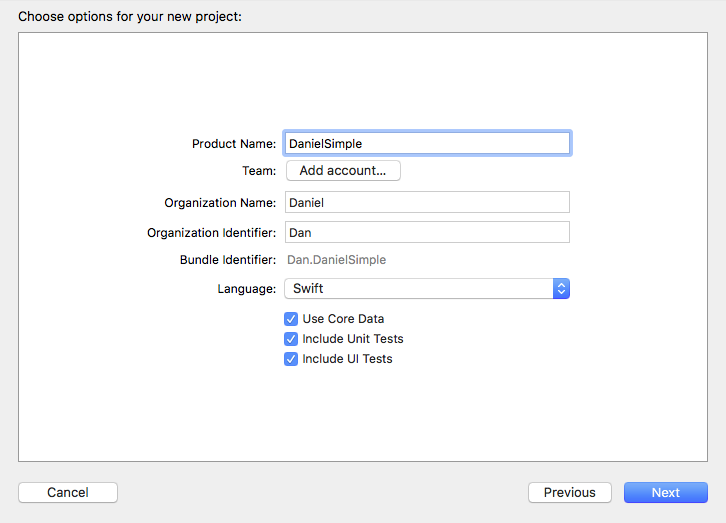
## Xcode Overview

Before we start doing anything, it is useful to familiarise yourself with the Xcode environment.

1. Start Xcode, then select File > New > Project

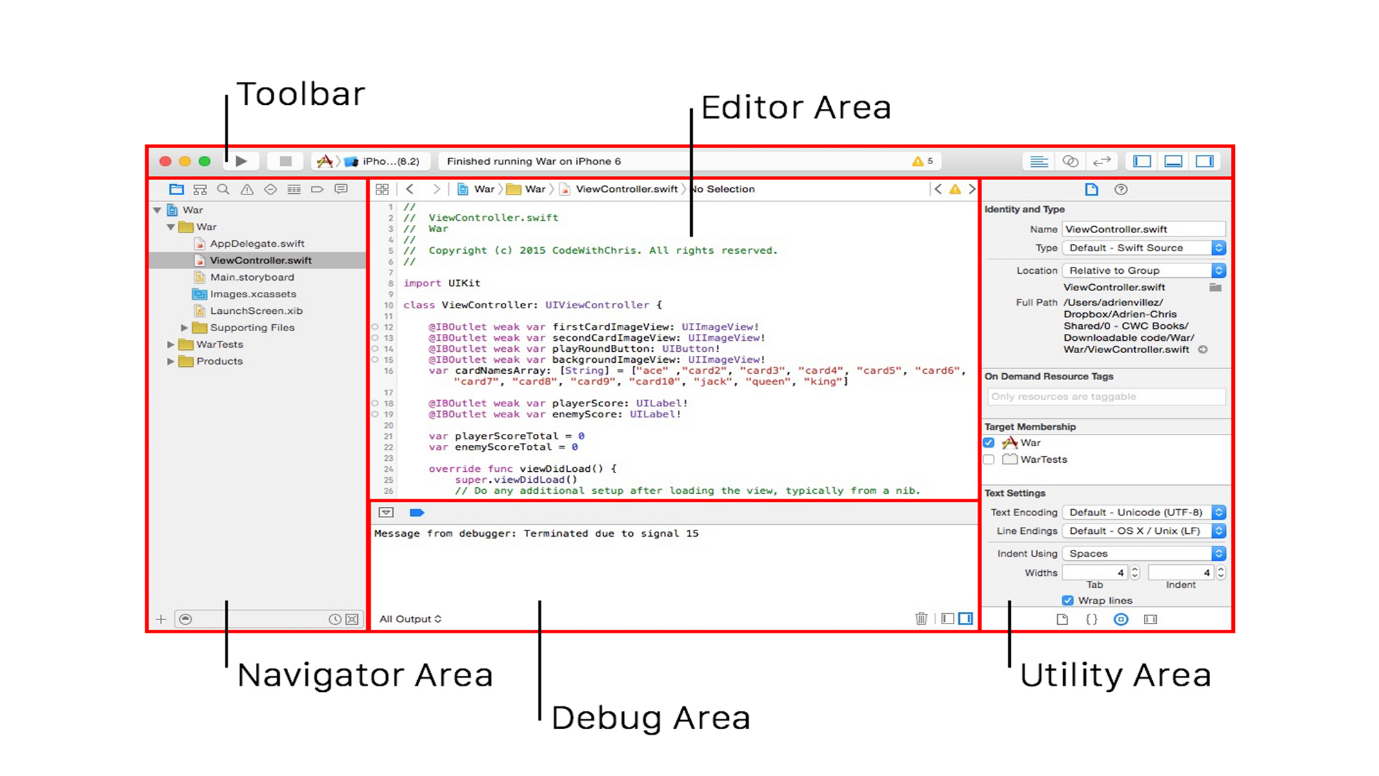


1. Select ‘Single View App’



1. Fill in Product name, organisation name, organisation identifier, and make sure language set to Swift
2. Create a New Folder (on desktop or student drive as appropriate), and create

Xcode has four main areas and a toolbar

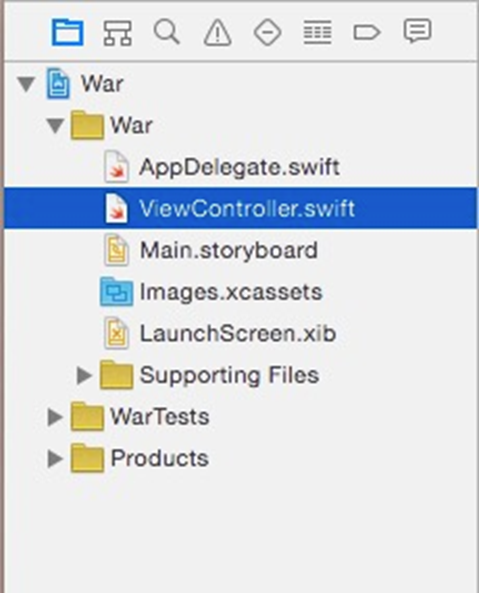


The sizes of the different areas can be adjusted by dragging, and all areas apart from the Editor Area can be hidden using these buttons:

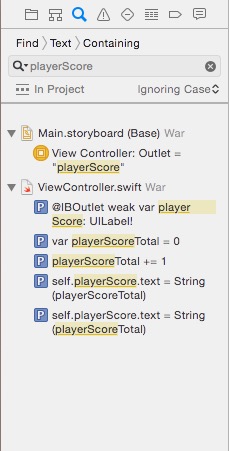


### Navigation area

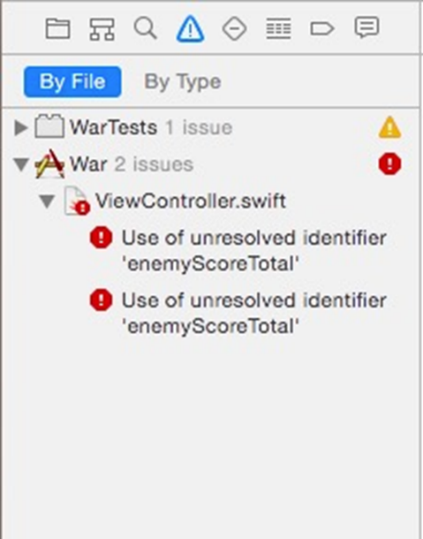
The Navigation Area is where all your project files are shown. You must have the folder icon at the top highlighted to browse files. The two most important files are .storyboard files are where your user interfaces are built and .swift files which contain your code.



You can search the files for keywords. By selecting the magnifying glass

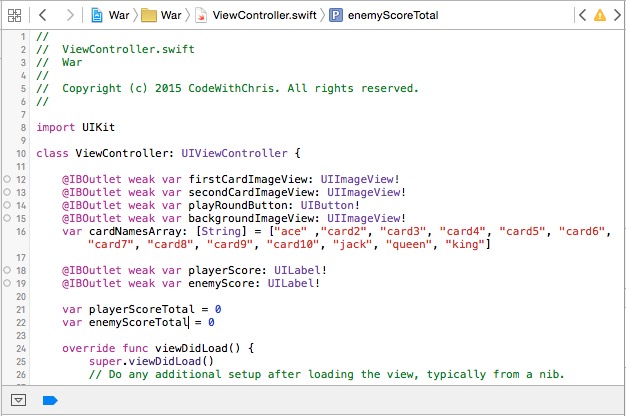


You can look through issues using the warning triangle

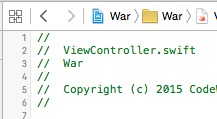


### Editor Area

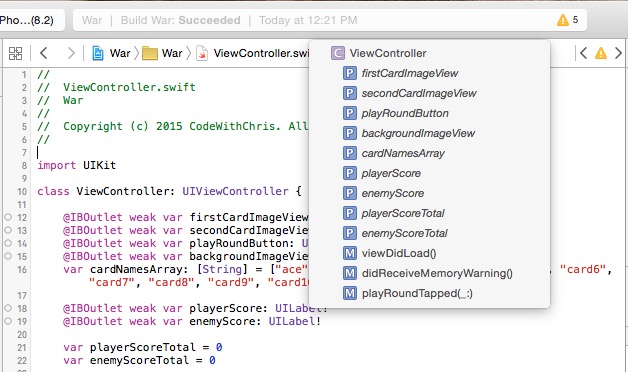
This is where you will spend most of your time. If you have a .storyboard file open then you will get the “drag and drop” style UI Builder If you have a .swift file open, you will have access to the underlying code.



Navigation between files is provided in the top section. The back and forward buttons are very useful. This navigates through the levels of the file structure. The “Jump Bar” also allows you move between files quickly



In a .swift file, clicking on the last section in the Jump Bar, you will get a list of methods in the file, clicking on one of those will take you to that method in the file



There are three types of Editor View:

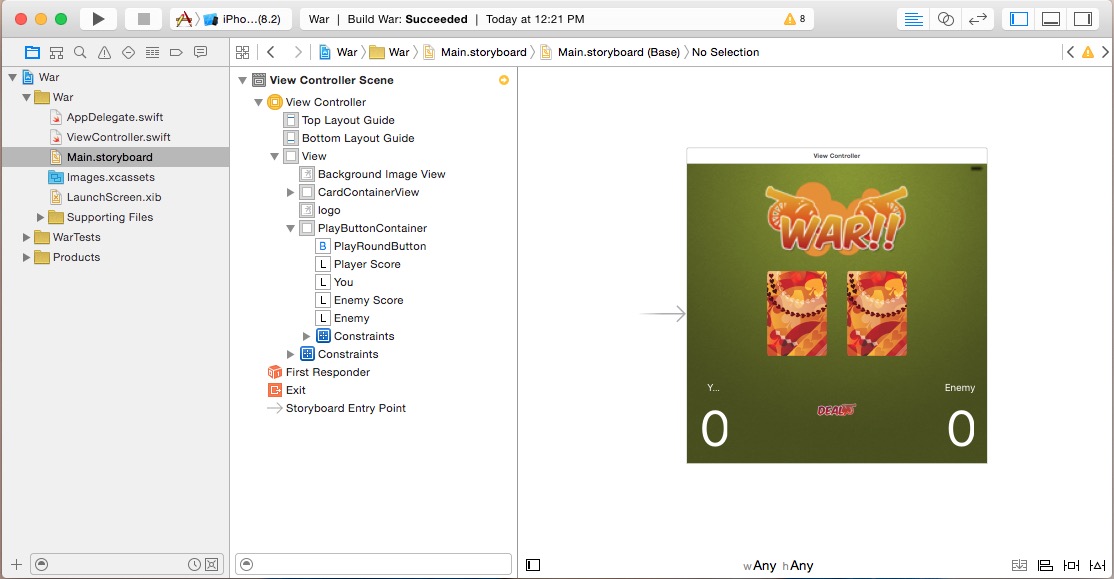
* **Standard Editor -** Single screen showing just this file
* **Assistant Editor -** Shows an accompanying file to the file you are currently looking at. Each pane has independent jump bars.
* **Version Editor -** Used when looking are source controlled code.

You can switch between these using these icons:



### Interface Builder

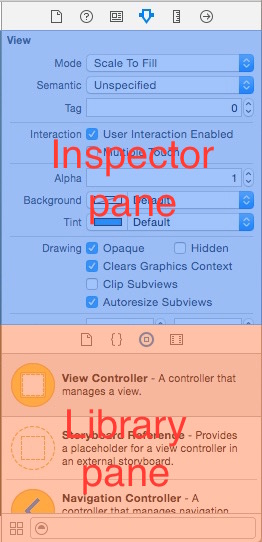
This is useful for building your Apps user interface. It has a separate navigation pane for navigating the hierarchy of your UI objects.



### Utility Area

The Utility area is comprised of two panes, the inspector pane and the library pane. (in newer xcodes the library is the icon in the top next to the editor area toggles)

Utility Area is mainly used in the .storyboard files. It will have all the information about the interface object you have selected.



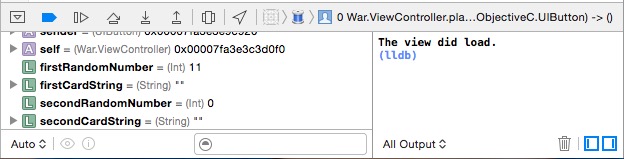
The Subsections of the utility pane are (at top, from left to right)

* **File Inspector -** Information about the file (e.g. location, type)
* **Quick Help Inspector -** Gives you information from Apple documentation about your selected item
* **Identity Inspector -** Tells you about the class for the UI Element, also Accessibility options.
* **Attributes Inspector -** (Very important) Varies depending on what you have selected, allows you to change the elements attributes, e.g. for a text input box, you can change the keyboard type.
* **Size Inspector -** Edit constraints, edit size
* **Connections Inspector** Shows what the element is connected to in the code.

### The debug area

The debug area will show you console output and the state of various variables when you run your application.

The icons along the top are Play/pause, Step Over, Step Into, Step Out, Debug Hierarchy, Simulate Location



The console on the right will display output and any content written to the console using print(“”)

### The toolbar

Run/ stop your app. Toggle views. See build results and any warnings.



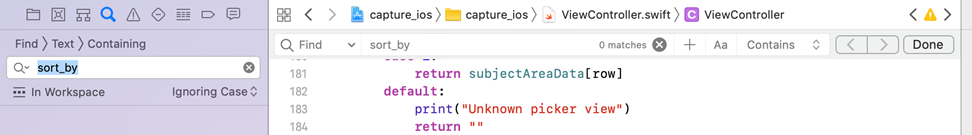
### Useful stuff

Finding text

CMD+F search this file

CMD+Shift+F search whole project

Click on find, to change to replace



Jumping to definitions

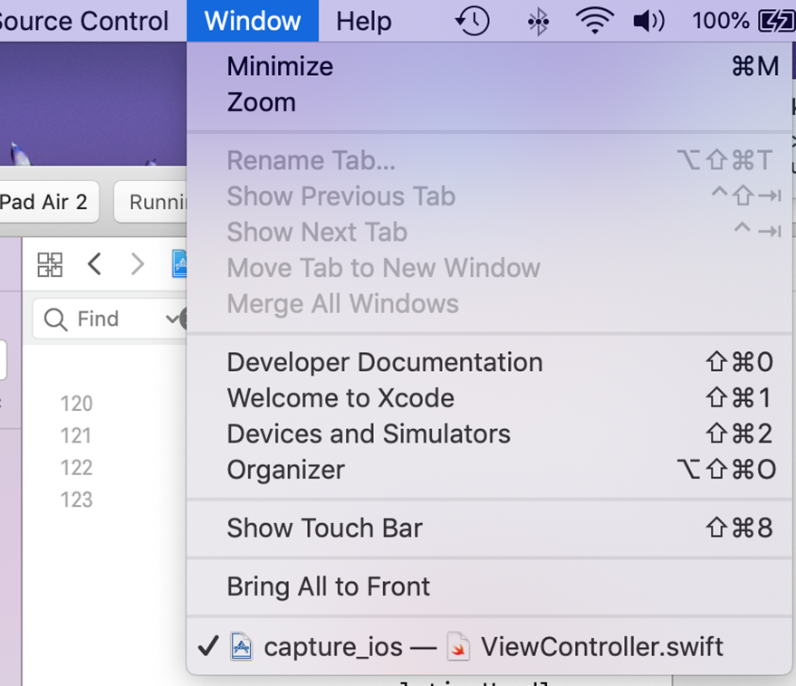
CMD+ Left click. To go to definition of method or field

### Window Menu

Developer documentation, very useful.

Manage devices and simulators.

Organizer, shows history of app distribution builds. If your app is on the app store, it will show crashes and energy logs



### Swift Coding standards

<https://swift.org/documentation/api-design-guidelines/>

There are too many to go through in detail, here are some important ones:

* Try to make sure you code compiles without warnings
* Using argument (parameter) labels helps understanding of a function
* Prefer method and function names that are grammatically correct, along with the argument labels
* Avoid Abbreviations, unless they can be easily searched for and found
* Follow case conventions. Names of types and protocols are UpperCamelCase. Everything else is lowerCamelCase
* Swift is Object-Oriented so function overriding / overloading is all fine

## Three Simple Examples

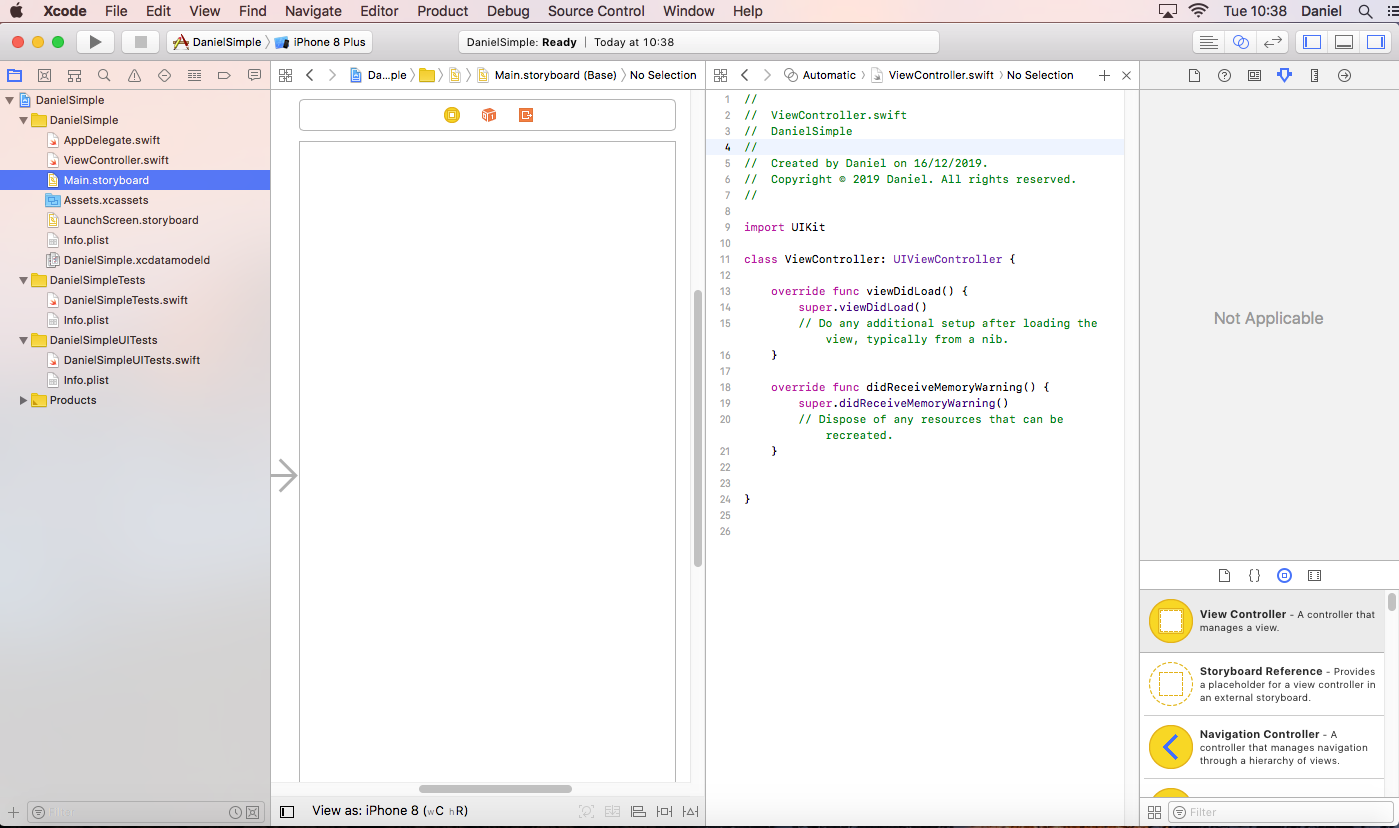
Typically, we will be mainly working with two files when developing our applications. The Main.storyboard is a drag-and-drop graphical interface builder. The ViewController.swift contains the Swift code operating behind the interface. An interface Object, such as a button, can have Actions associated with it. These Actions are fired by different interface events, such as a click. An interface Object also has properties, known as Outlets, which can be changed, for example, clicking on a button could change the text displayed in a label. The Actions and Outlets are defined in the Swift code.

We will look at three simple examples to see how this works and to get a feel for Xcode. The simple button example displays a text label which changes when a button is clicked. The simple text entry example allows the user to type in text which is displayed in a label. The simple timer example uses a timer to run a counter.

### **Simple button example**

Our first simple example is a text label which changes when a button is clicked.

In Xcode you should have view shown below, with Main.storyboard on left Edit Pane and ViewController.swift on the right Edit Pane. If not then go to View > Assistant Editor – check “Assistant Editors on Right” and “Show Assistant Editor”. If you have the View Controller Scene displayed with the storyboard in the left Edit Pane, you can close this using the hide menu button at the bottom of the pane.



The available interface objects are shown in the Library Pane in the bottom right-hand corner.

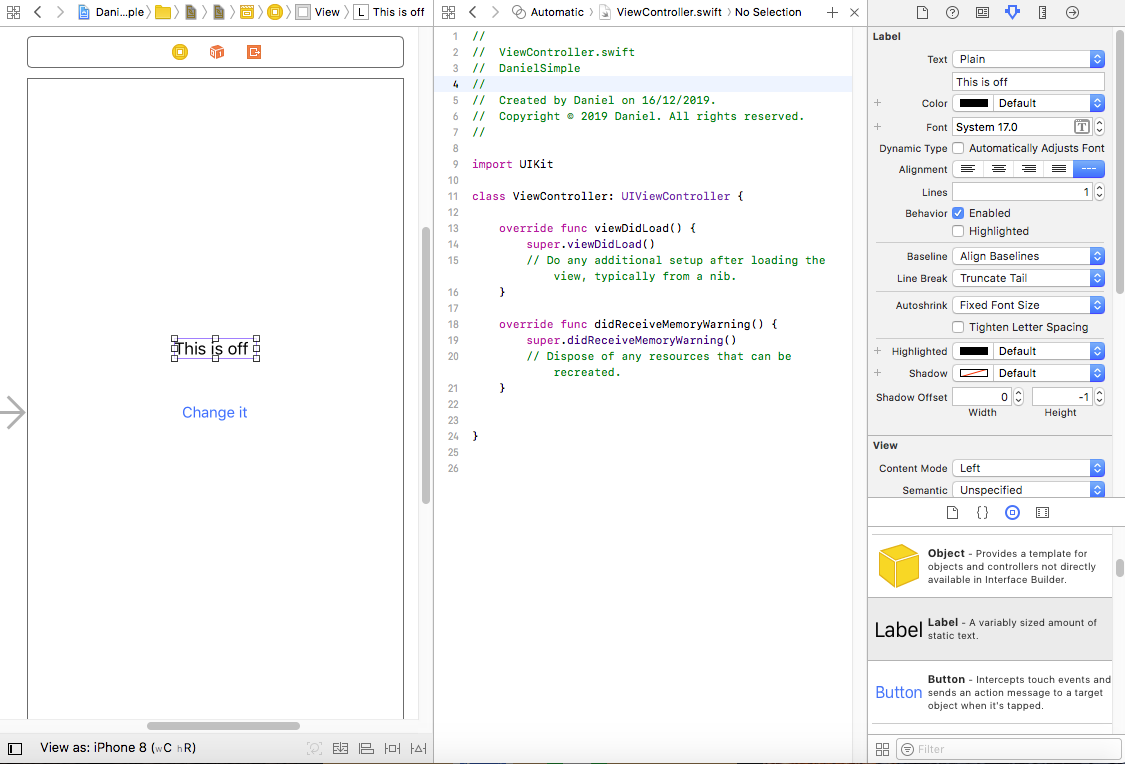
1. Drag a ‘Button’ from the Library Pane onto the storyboard
2. Check that you are in in Attributes view in the Inspector Pane (top right-hand corner)
3. 
4. Change the text of the button (in the text box below “Title”) to “Change it”
5. Resize the button on the storyboard to make the text fit
6. Run the simulator by clicking the play button (top left-hand corner), you may need to click on simulator icon to bring it to the front



This shows the interface on a simulated i-phone. At the moment it doesn’t really do anything, though you can click on the button. Stop the simulation

Next we will add the label.

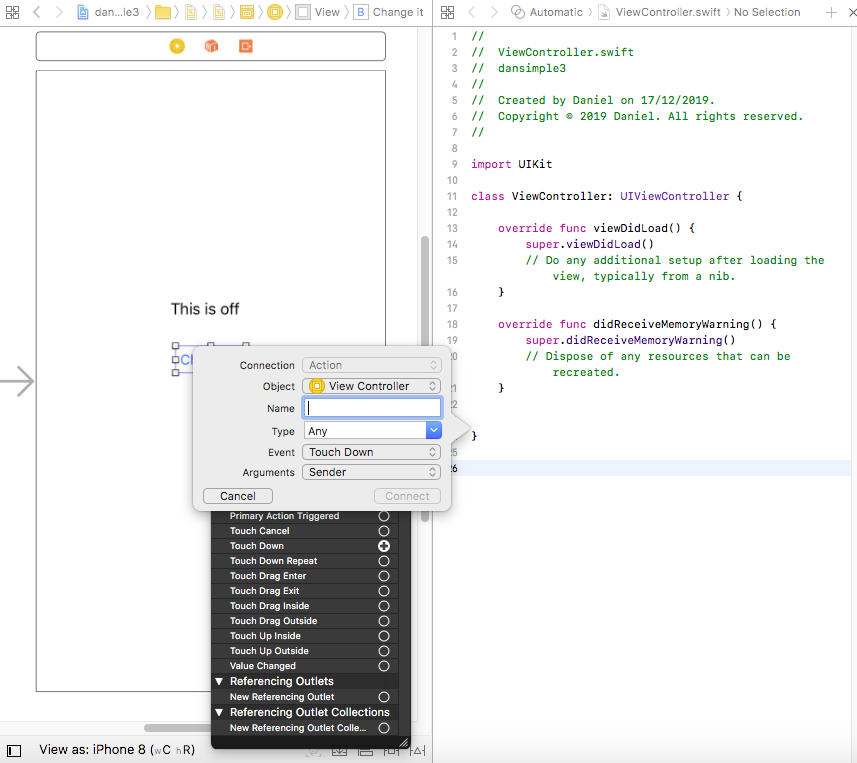
1. Drag a ‘Label’ from the Library Pane, on to the storyboard.
2. Change the text of the label (text box below “Text”) to “This is off”
3. Resize the label on the storyboard to make the text fit



1. Run the simulator again, try clicking the button. Nothing happens – we have a button and a label, but no functionality, because we haven’t written any code yet. Stop the simulation

Next, we will add an Action to the button.

1. Right click on the button.
2. Left click and hold on “Touch Down” and drag the link to the Swift code inside the ViewController class.



1. Name it “buttonWasTapped”.
2. Click connect.

This Inserts new code stub for a function called buttonWasTapped into the ViewController class



When there is a Touch Down performed on the button, this function will be executed.

However, at the moment we cannot reference the Label object, as we need to connect this too.

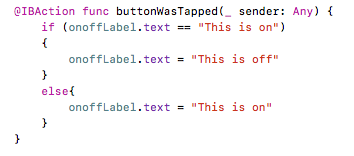
1. Right click on Label
2. Left click on “New Referencing Outlet”
3. Drag to ViewController.
4. Name it “onoffLabel”

This adds a new Outlet reference into ViewController



We can now refer to the Label object from within the swift code.

1. Edit the buttonWasTapped function as follows



1. Run the simulator. Now when we click on the button, the label will change.

### **Simple text entry example**

Our second simple example allows the user to type in text which is displayed in a label.

You can add this new functionality to your existing application, you do not need to create a new one.

1. Drag a ‘Text Field’ from the Library Pane, on to the storyboard.
2. Change the text of the text field (text box below “Text”) to “keyin”
3. Right click on the text field.
4. Left click and hold on “Did End on Exit” and drag the link to the Swift code inside the ViewController class.
5. Name it “doneEditing”.
6. Click connect.

This Inserts new code stub for a function called doneEditing into the ViewController class. When there is a ‘Did End on Exit’ (triggered by user entering return) performed on the text field, this function will be executed.

Next we will add the label that will be changed when the user enters text.

1. Drag a ‘Label’ from the Library Pane, on to the storyboard.
2. Change the text of the label (text box below “Text”) to “Label waiting”
3. Resize the label on the storyboard to make the text fit

We now need to create Outlet references for both the text field and the label.

1. Right click on Label
2. Left click on “New Referencing Outlet”
3. Drag to ViewController.
4. Name it “waiting”
5. Right click on Text Field
6. Left click on “New Referencing Outlet”
7. Drag to ViewController.
8. Name it “keyin”



Now all we need to do is to add some functionality to the code stub function called doneEditing.

1. Edit the doneEditing function as follows

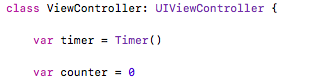


1. Run the simulator.
2. Check under Hardware > Keyboard that ‘Connect Hardware Keyboard’ is NOT checked. We want to use the on-phone keyboard, not the Mac keyboard.
3. Now when we type text into the text field and press return, the label will change.

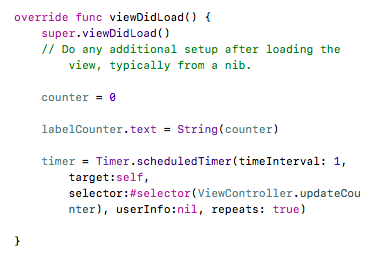
### Simple Timer Example

In this example, we are going to create a simple timer to run a counter. We will then add some additional functionality to control the counter.

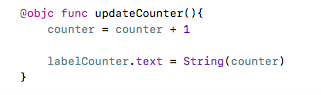
1. Start a new project – File > New > Project
2. Drag a ‘Label’ from the Library Pane, on to the storyboard
3. Change the text of the label (text box below “Text”) to “LabelCounter”
4. Right click on Label
5. Left click on “New Referencing Outlet”
6. Drag to ViewController
7. Name it “labelCounter”
8. Add the following variables to the ViewController class



1. Initialise variables and create the timer in the viewDidLoad function



1. Add an update function for the counter



1. Run the simulator

### Additional exercises:

Extend the counter application in the following ways.

**Add a Reset button**

This resets the counter to zero while it is running (it does not stop the counter running).

**Add a Start button**.

The counter starts at zero and is initially not running. It starts running when the user clicks the button.

Hint: move the timer initiation code to a new function attached to Touch Down on a new button

**Change the Start button to a Start/Stop button.**

Initially, the counter starts at zero and isn’t running. Button shows ‘Start’

User clicks button, counter starts running, button shows ‘Stop’.

User clicks button, counter stops running, doesn’t reset to zero, button shows ‘Start’.

Change button text using e.g.: runButton.setTitle(“Start”, for: .normal)

Stop timer using: timer.invalidate()

Hint: use a variable to store whether counter is running or not