# Introduction to Table Views

Table views are one of the key elements in a mobile device interface. Table views allow us to display a list of data, often this will be a list of options that the user can select from.

This workbook includes three examples. The first example is a simple list of text items. The second example is a simple list in which each item is an image and text. The third example is a rudimentary To Do list application. It allows the user to add items to the list and to delete items from the list.

This workbook 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. It also assumes you have completed the previous workbooks.

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>

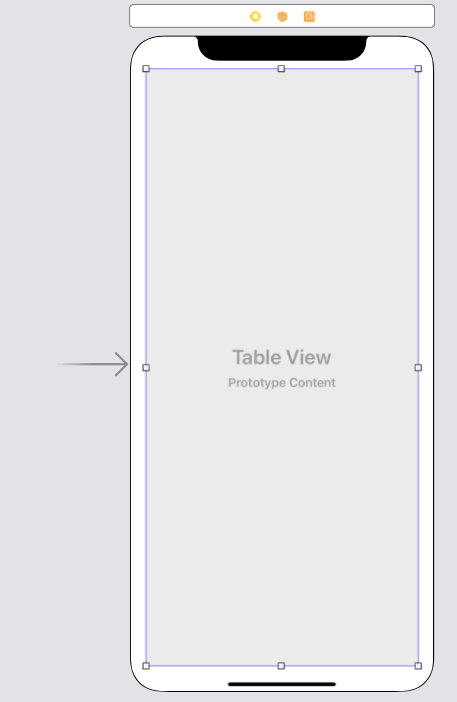
The codeacademy website has a course on Swift, but it is new and doesn’t contain many lessons at the moment. Our library also has books on Swift.

There are different versions of Xcode and Swift, so watch out for incompatibilities!

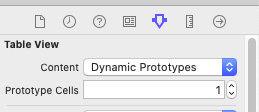
## Basic List Example

In this example we will just make a basic list of text items.

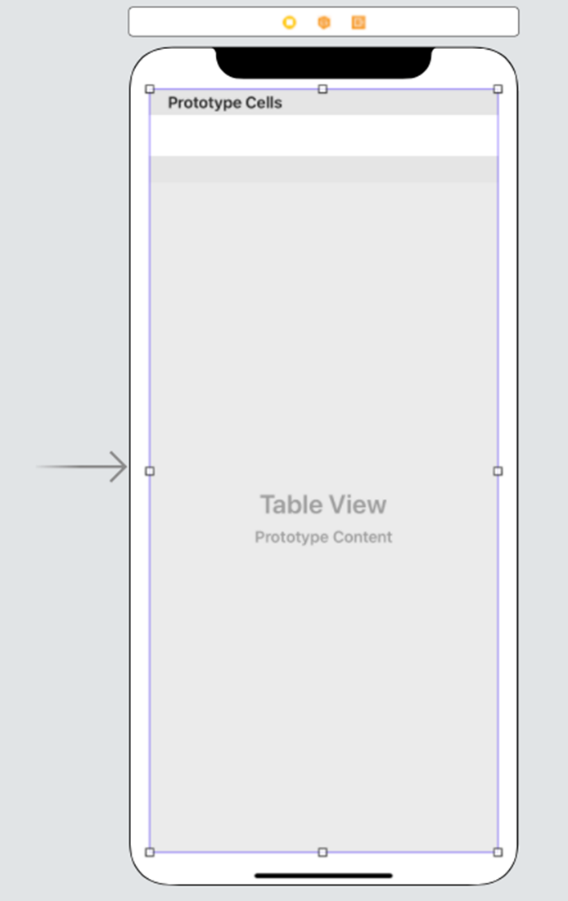
1. Create a new Single View Application
2. Add a Table View from the Library
3. Enlarge the Table View to fill the scene



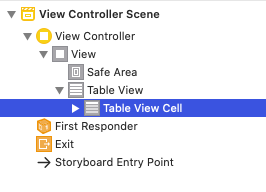
1. Click on the Table View
2. In the Attributes pane, check that Content is 'Dynamic Prototypes'



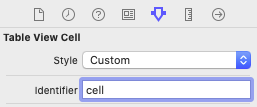
1. Change Prototype Cells to one



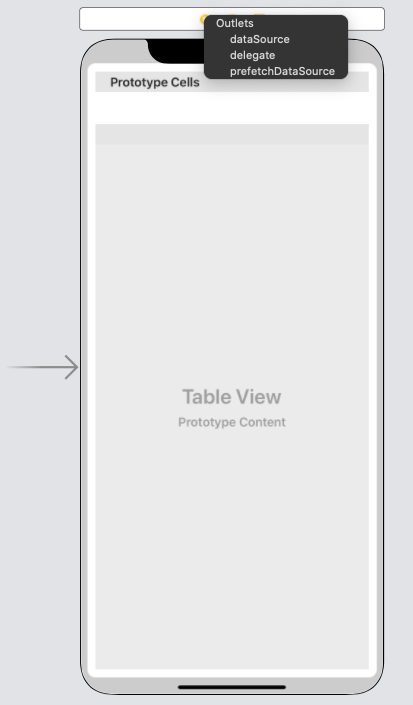
1. Click on Table View Cell in the View Controller Scene navigator



1. In the Attributes pane, change the Identifier to cell



1. Ctrl-click on the Table View and drag to the View Controller icon (left-hand icon) and select datasource from the pop-up



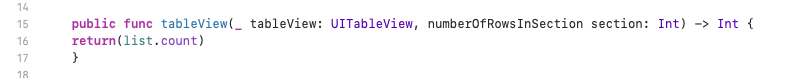
1. Ctrl-click on the Table View again and drag to the View Controller icon and select delegate from the pop-up
2. Open ViewController.swift
3. Add additional delegates to the ViewController class



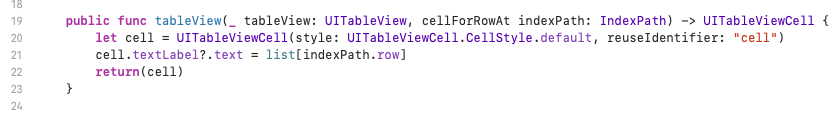
1. In the View Controller class, add a list of list items



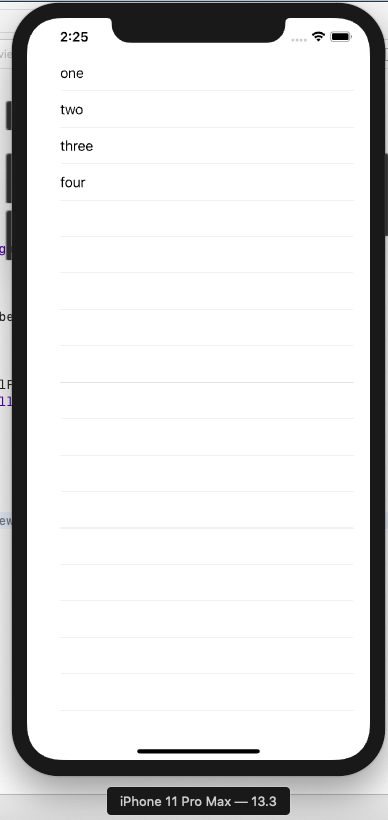
1. In the View Controller class, add a new function that returns the number of items in the list



1. In the View Controller class, add a new function that will populate the table



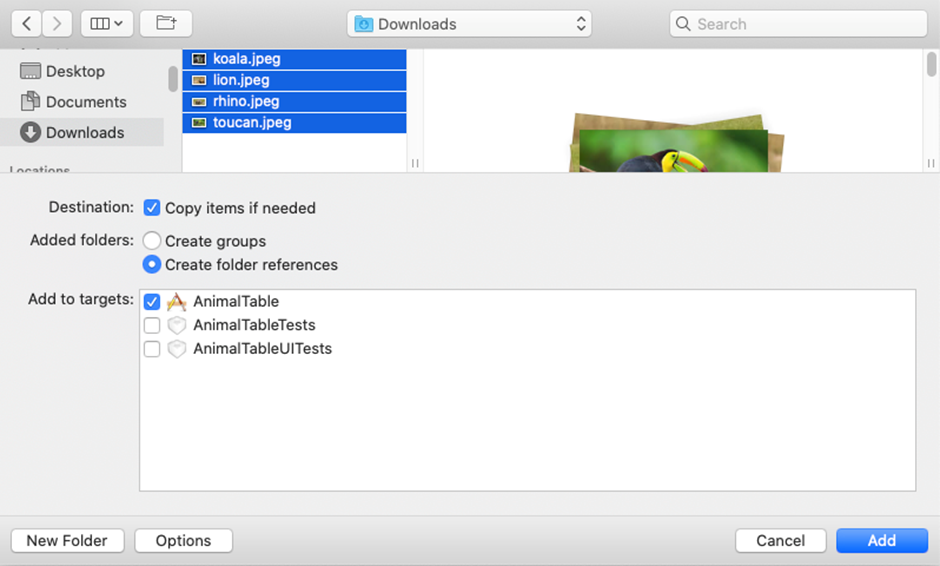
1. Run it in the simulator, it should look like this:

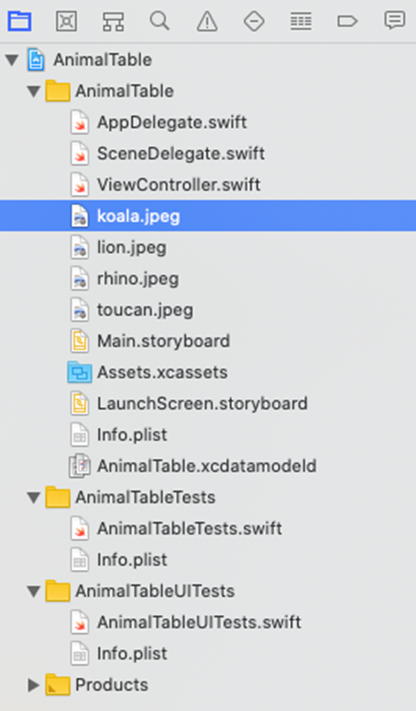


## Another Simple List Example

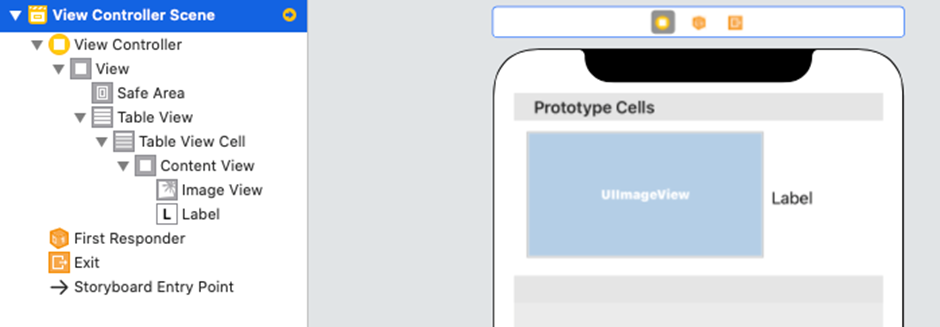
This time we are going to make a slightly more interesting list, where each item contains an image as well as text.

1. Create a new Single View Application
2. Download four animal images from the internet; make sure they are all the same format, e.g. jpeg.
3. Using either File > Add Files to… or drag and drop, add the files to the project



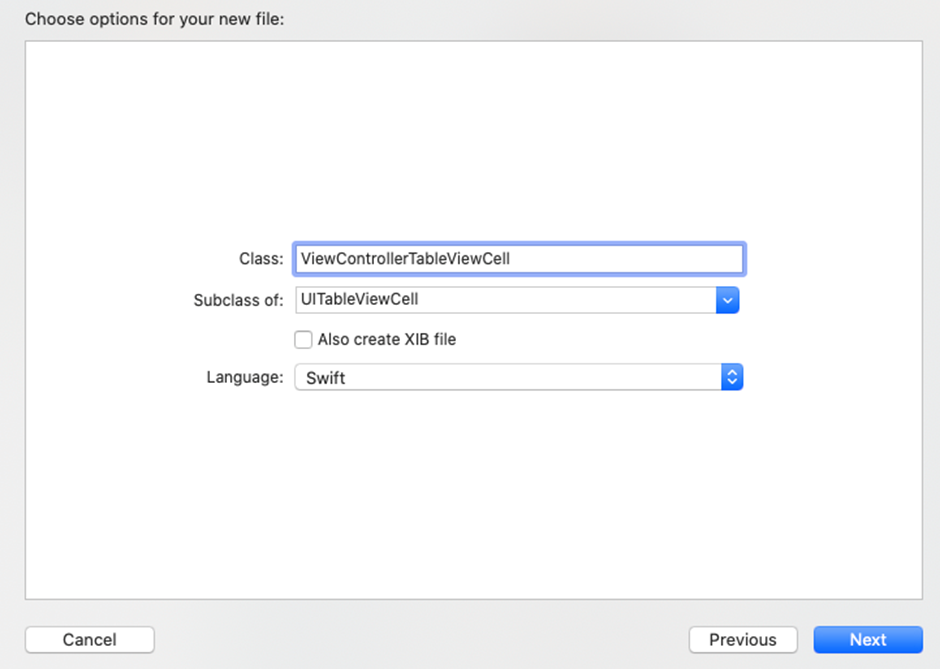


1. Add a Table View from the Library
2. Enlarge the Table View to fill the scene
3. Click on the Table View
4. In the Attributes pane, check that Content is 'Dynamic Prototypes'
5. Change Prototype Cells to one
6. Click on Table View Cell in the View Controller Scene navigator
7. In the Attributes pane, change the Identifier to cell
8. Ctrl-click on the Table View and drag to the View Controller icon (left-hand icon) and select datasource from the pop-up
9. Ctrl-click on the Table View again and drag to the View Controller icon and select delegate from the pop-up
10. From the library, drag an Image View into the prototype cell.
11. From the library drag a Label into the prototype cell
12. You may need to adjust the size of the image and/or the size of the cell to make it all fit

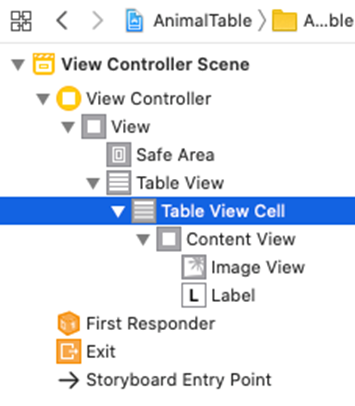


We now need to create a new View Controller to control the view of each cell.

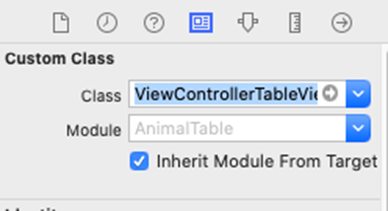
1. File > New > File…
2. Cocoa Touch Class
3. Set the class equal to ViewControllerTableViewCell and create



1. Click on the cell in the View Controller Scene Navigator



1. **In the Identity Inspector pane, change the class to our new ViewControllerTableViewCell**



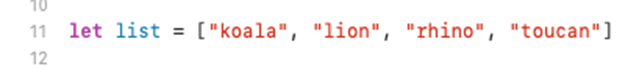
1. Open ViewControllerTableViewCell.Swift from the navigation pane
2. Create an Outlet from the imageView, named myImage
3. Create an Outlet from the label, named myLabel



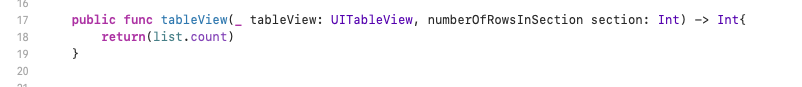
1. Now open the ViewController.Swift file
2. Add additional delegates to the ViewController class



1. In the View Controller class, add a list of animal names (the names must match the name of your files)



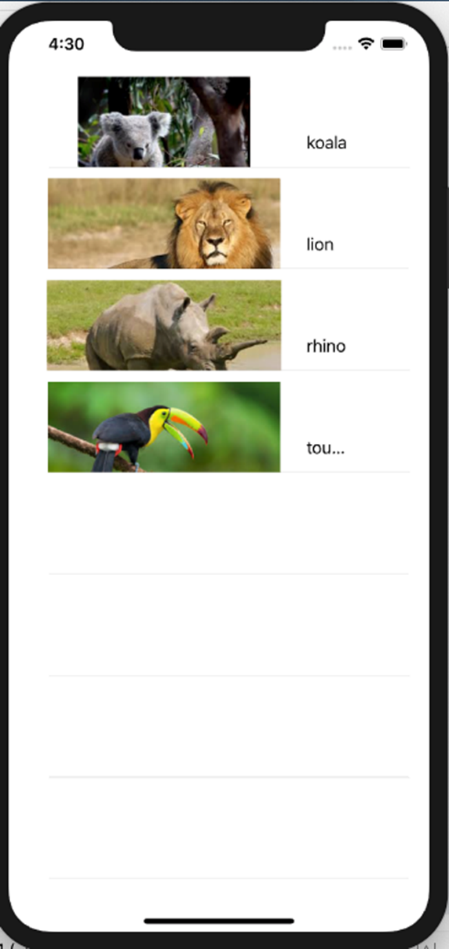
1. Add a function to return the number of rows needed for the table



1. Add a function to populate the table



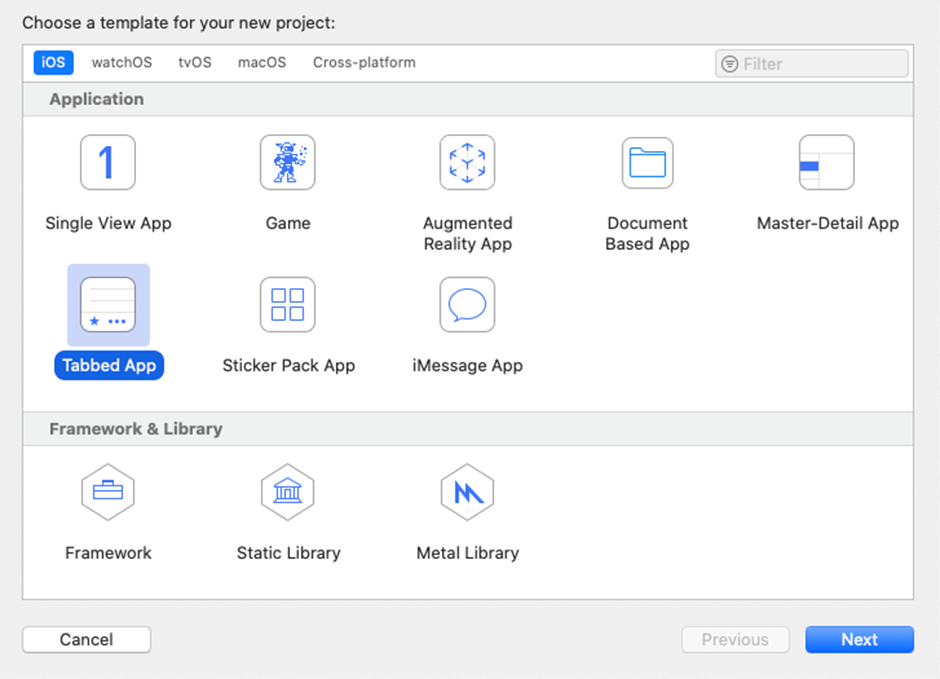
1. Run the simulator, it should look something like this (I need to make my labels longer and find a better proportioned picture of a koala):

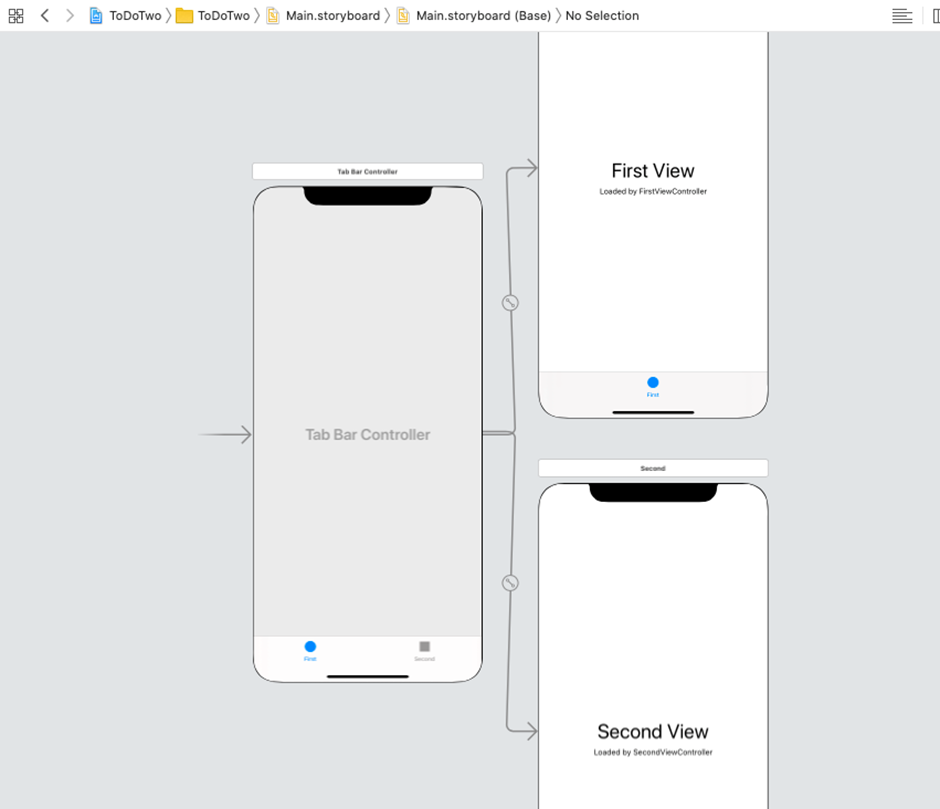


## To Do List Example

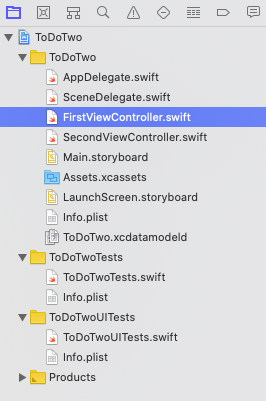
In this example, we will make a basic To Do application. The application will allow the user to add and delete items from the list. The list and the functionality for adding items will be on different tabs of the application.

1. Create a new **Tabbed** Application

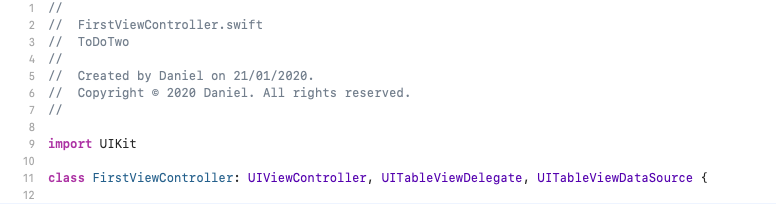




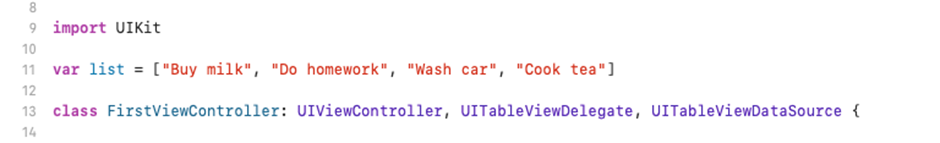
1. Add a Table View from the Library to the First View tab
2. Enlarge the Table View to fill the scene
3. Click on the Table View
4. In the Attributes pane, check that Content is 'Dynamic Prototypes'
5. Change Prototype Cells to one
6. Click on Table View Cell in the View Controller Scene navigator
7. In the Attributes pane, change the Identifier to cell
8. Ctrl-click on the Table View and drag to the View Controller icon (left-hand icon) and select datasource from the pop-up
9. Ctrl-click on the Table View again and drag to the View Controller icon and select delegate from the pop-up
10. In FirstViewController.Swift…



1. Add the delegate and source to the FirstViewController class



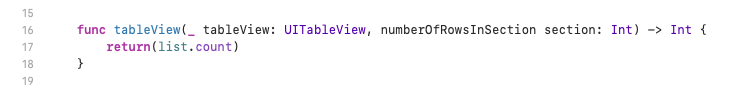
1. Add a list of tasks to do – we will want to access this list from the second tab, so we need to make it a global variable. We do this by writing it outside the class declaration:



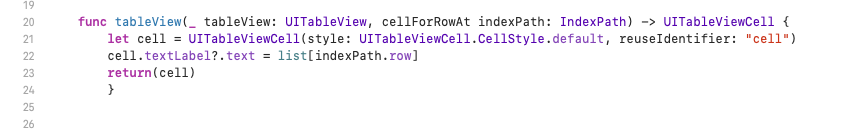
1. add an outlet from the table called myTableView



1. Add code to return number of rows:



1. Add code to populate the table



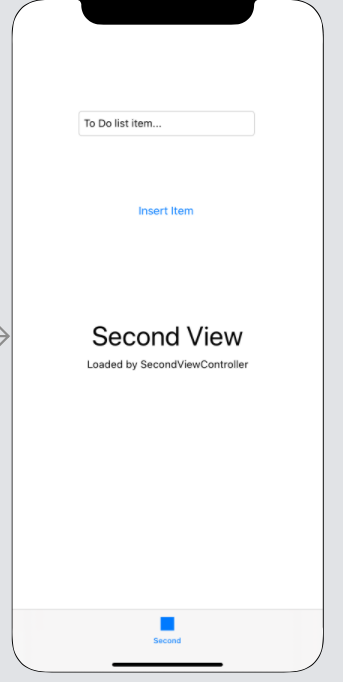
1. We will now add some code to delete items from the list using a left swipe:



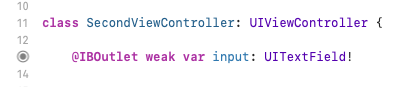
1. Run the simulator; try left swiping an item with the mouse to delete it.

We will now add some functionality on the second tab that will allow us to add new items to the To Do list. The user will type in the new item, then click on a button to add it to the list.

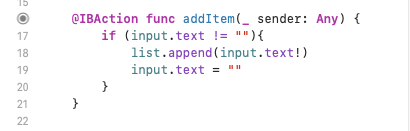
1. Add a text box to the second view
2. Add a button to the second view



1. In SecondViewController.Swift, add an outlet called input from the text box



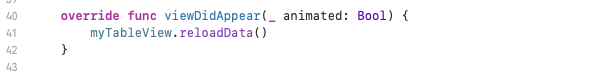
1. Add a 'touch down' action to the button and add the following code to it



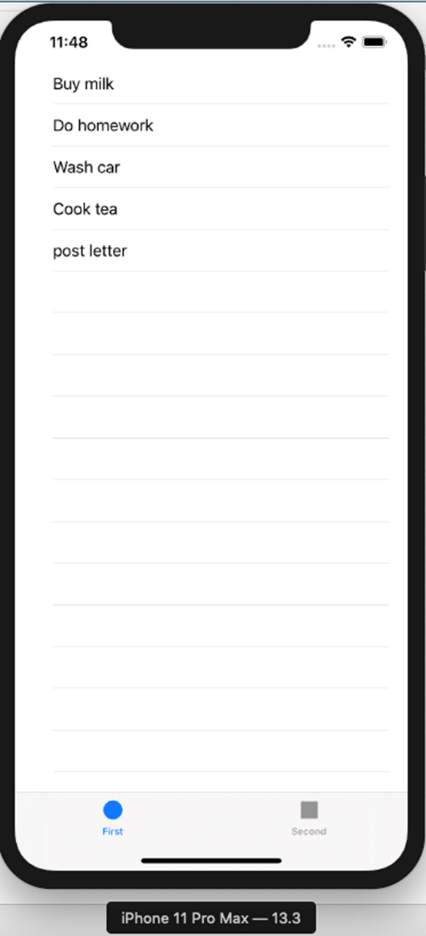
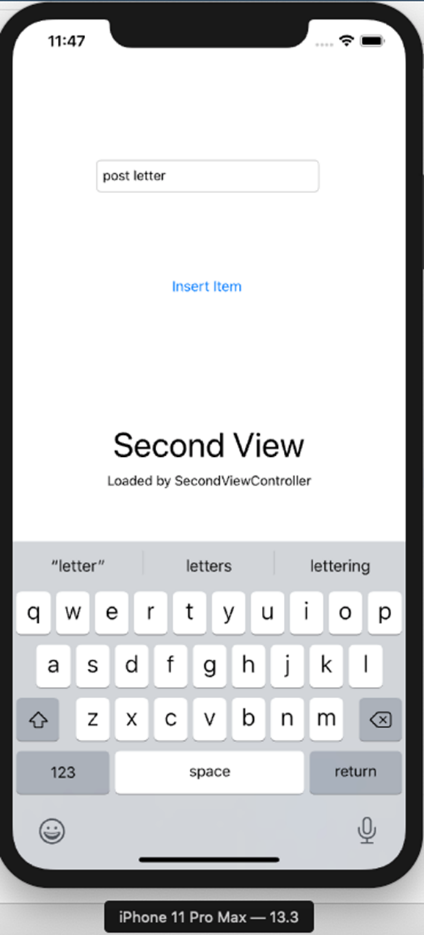
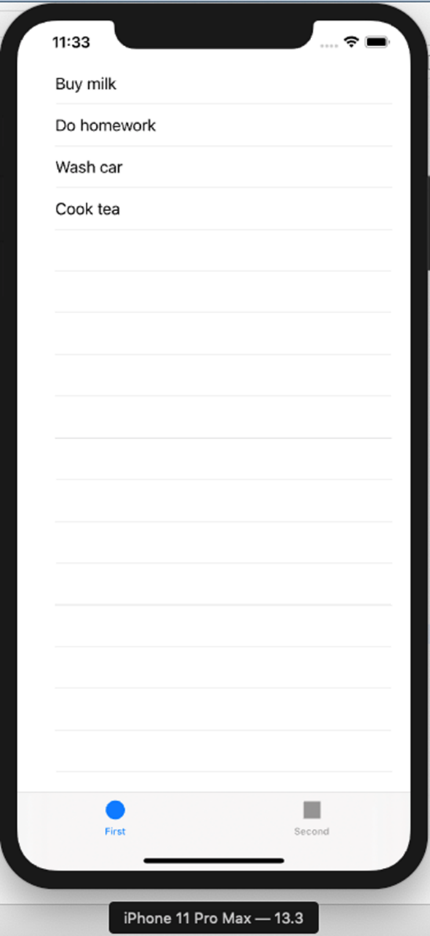
1. We need to close the on-screen keyboard when the user clicks return - add a 'Did End on Exit' action to text field, called endInput, and add the following code:



1. We need to add a list display refresh when we return to the first tab. In FirstView Controller, add the following function

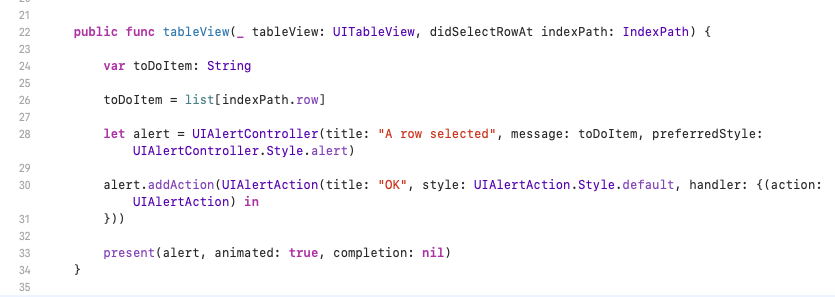


1. Run the simulator, remember to turn off hardware keyboard



Finally, we will add a bit of code to allow the user to select a cell from the table.

1. In FirstViewController add the following code



1. Run the simulator

