framework training
We love technology

Building iOS Apps in Swift

Course notes for part 1

14th-16th September 2021

020 3137 3920 @FrameworkTrain

frameworktraining.co.uk

Contents List

Day 1 - 14th September 2021

Day 2 - 15th September 2021

Day 3 - 16th September 2021

Session 1: Introduction to Swift

- Trainer and Delegate Introductions / Intro to course
- Foundations of Swift
- XCode Playgrounds
- Basics of the language: Types, Operators, Conditionals, Iterators, Strings, Arrays, Dictionaries
- Lab 1: Trying out the language basics

Welcome & Introductions

Chris Price

cjp@aber.ac.uk

framework training
business value through education

Building iOS Apps in Swift

- 6 day course in two parts
- Based around excellent material provided by Apple
- Speeded up for professionals with experience
- Expanded on in complex areas not covered well by Apple's course

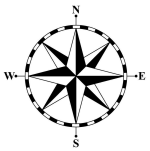
Introductions - me

- Course Instructor: Chris Price
  - Professor at Aberystwyth University, Wales
  - Teach courses in Software Engineering and in App Development in Swift
  - Been developing apps since 2009
  - Shipped apps to around 200,000 users in that time

Introductions - you

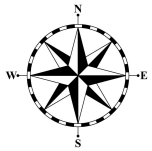
- Course attendees
  - You…
    - Name
    - Professional Position
    - Programming experience
    - Motivation and Expectations

Course Content



- Part 1: First three days
  - Basics of Swift language (briefly)
  - Extra features of Swift needed for app building
  - Enough about building iOS apps to build *most* self-contained apps
- Part 2: Second three days - more advanced apps
  - Persistence
  - Linking to internet systems, cloud services, libraries
  - Notifications
  - Reactive systems and concurrency
  - Automated testing

How will the course work



- We will cover basics of Swift very quickly, and try things out in Swift Playgrounds
- We will take more time over possibly unfamiliar features such as Optionals, Protocols and Closures
- We will build *many* small apps in Xcode, some together and some following instructions in Apple Books
- Support each other in building, and I will help where needed

Questions/comments?

?

Introduction to Swift and Playgrounds

Chris Price

Swift - a modern language: Safe

- Strong typing
- Compile-time checking as much as possible
- Ensures that things are initialised
- Makes switch statements cover all possible cases
- Makes clear you know what is included in an if statement
- Takes nil pointers seriously

Swift - a modern language: Fast

- Language that helps compiler to optimise
- Encourages the user to make their intentions clear so that compiler can optimise the code

Swift - a modern language: Expressive

- Doesn't make people write stuff the compiler should know:
- Implied type declaration where possible
- Implicit type name when type known (e.g. for enums)

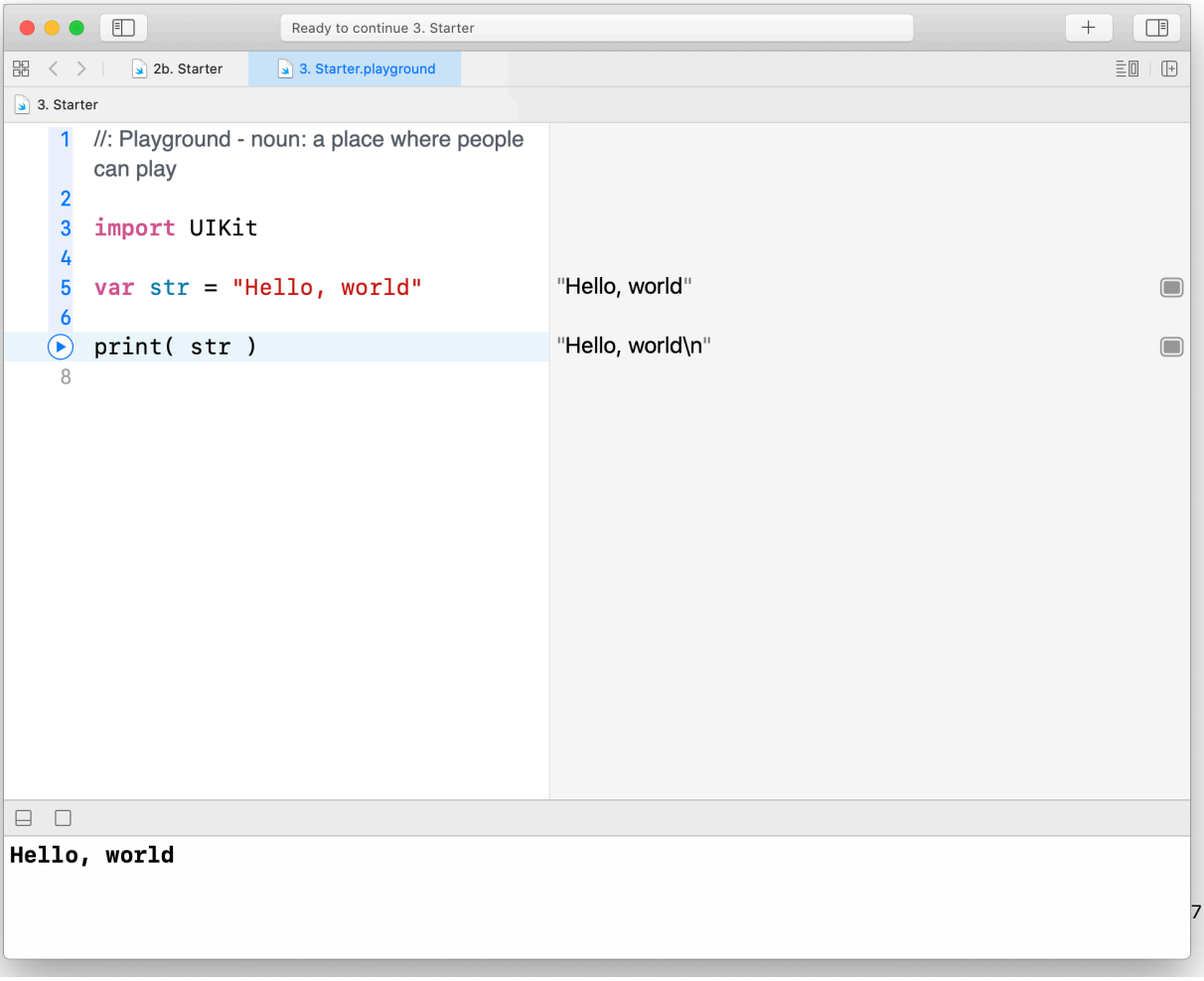
- Has the features you might expect in a modern language
- Powerful Collections
- Protocols
- Extensions
- Functional programming
- Ints, Doubles etc are first class items

Hello, world

```
print("Hello, world!")
```


Try making a playground

1. Open Xcode
2. Choose File > New > Playground
3. Select iOS, select the Blank template and click Next
4. Change str to "Hello, world!"
5. Hold cursor over line you want to compile to and a blue circle with triangle in it will show
6. Click on blue triangle to run all code up until there



7

Explore playgrounds and language basics

1. Try running "Starter Playground" included in Session 1 of the course
2. See how Playgrounds shows you the run values
3. There is a 4 page summary of the basics of the language included - try cutting and pasting language examples into a Playground, and see what they do.

More on material in this session

- We have very briefly covered the basics of the language
- They are covered much more slowly in Develop in Swift Fundamentals:
  - Lesson 1.1: Swift Intro and Playgrounds
  - Lesson 1.2: Constants, variables, types
  - Lesson 1.3: Operators
  - Lesson 1.4: Control flow
  - Lesson 2.1: Strings
  - Lesson 2.5: Collections
  - Lesson 2.6: Loops

9

We will look in more detail at:

- Lesson 2.2: Functions
- Lesson 2.3: Structures
- Lesson 2.4: Classes
- Unit 3: Optionals / Guard / Scopes

10

The Swift Language - Brief Overview (1)

Chris Price v1.4, Sept 2019

Simple Variable Types

Int, Double, Bool

Can be implicitly declared:
let x = 42 // Int constant
var y = 3.1 // Double variable

Explicit declaration:
let z: Int = 42 // Int constant
let a: Double = 3 // Double constant
let cold: Bool = true //Boolean constant

Common Operators

Arithmetic Operators

+	Add	// e.g. y = a+b
–	Subtract	
*	Multiply	
/	Divide	
%	Remainder	

Arithmetic shorthand

// e.g. y += 21 same as y = y+21

+=	Add and assign
–=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Remainder and assign

Unary operators

-	Minus	//e.g. y = -x Gives negative of a number

Boolean tests – return true or false

<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&&	Logical AND
||	Logical OR

Conditionals

```
if comparison1 {
 //called if comparison 1 is true
}
else if comparison2 {
   //called if comparison 2 is true
}
else {
   //called if nothing else is true
}

e.g.
let age = 27
if age >= 18 {
   print( "allowed to vote")
} else {
   print( "not old enough to vote")
}
```

Curly brackets are mandatory.
Round brackets around condition are optional.

Switch Statement

```
let age = 1
switch age {
case 0,1:
   print( "baby" )
case 2...4:
   print( "toddler" )
case 5..<18:
   print( "child" )
default:
   print( "adult" )
}
```

First applicable case is executed then exits. Must have a default unless all values are accounted for by other cases. Can switch on more than one value.

```
let num = 15
switch (num%3, num%5) {
   case (0, 0):
      print( "FizzBuzz" )
   case (0, _):
      print( "Fizz" )
   case (_, 0):
      print( "Buzz" )
   default:
      print( num )
}
```

The Swift Language - Brief Overview (2)

Chris Price v1.4, Sept 2019

Strings

Simple string expressions and printing out

```
let emptyString = ""
let greeting = "Hello"
let friendlyGreeting = greeting + ", friend"
print( friendlyGreeting ) // prints to console
```

String interpolation

```
let temp = "Boiling water is \(100*1.8 + 32)F"
```

String tests

```
if emptyString.isEmpty ...
if greeting.hasPrefix( "He" )...
if greeting.hasSuffix( "matey" )...
```

Equality tests whether made up of same characters in same order

```
if emptyString == ""     //this is true
if greeting == "hello"   // this is false as lower case
if "aardvark" < "apple"  // lexically less, so true
```

Useful string features

```
greeting.count // returns string length (5)
greeting.lowercased()  //returns "hello"
```

Manipulating characters in a string

```
let me = "Chris Price"
let space = me.firstIndex(of: " ") ?? me.endIndex
let name = me[..<space]
```

Arrays

```
//Declare empty array of strings
var myPets: [String]  =  [ ]
//Different way of doing same thing
var pets = [String]( )
myPets.count     // will be zero

myPets.append( "Chaz the Dog" )
myPets.append( "Dave the Goldfish" )
myPets.count    // will be two

//Declare constant array with one string in it
let yourPets = [ "Idris the Guinea Pig"]
// Combines the two string arrays into one string array
 var ourPets = myPets + yourPets
ourPets.count   // will be three

// Replace an element in a variable array
ourPets[0] = "Chaz the Bad Dog"
// Delete an element
let deadPet = ourPets.remove(at: 2)
ourPets.count    // now only two pets
// deadPet will contain the element removed

// can add extra element at specific position
ourPets.insert( "Bob the Capybara", at: 0)

// Initialise fixed size with default values
// Makes array with 200 values of 0.0
var readings = [Double]( repeating: 0.0, count: 200)
readings.removeLast()   // Can delete last element
readings.count              // will be 199
```

For Loops

for item in range { statements }

```
var sum = 0
for i in 1...5 { sum = sum + i }          // Gives the answer 15
sum = 0
for i in 1..<5 { sum = sum + i }          // Gives the answer 10
```

for item in collection { statements }

```
var listOfAttendees = [ "Bill", "Jane", "Jim", "Fred", "Ann" ]
for name in listOfAttendees{
   print( name )
}    // Will print each name in the list to console
```

Equivalent to 'for item in collection' using range

```
for i in 0..<listOfAttendees.count {
   print( listOfAttendees[i] )
}    // Will print each name in the list to console

// can get index and element by doing the following
for (index, name) in listOfAttendees.enumerated() {
   print( "\(index+1). \(name) " )
}
```

While Loops

while condition { statements }

e.g.

```
var i = 0
while (i < 100)
{
   print( "I told you so" )
   i++
}  // will print "I told you so 100 times
```

The Swift Language - Brief Overview (3)

Chris Price v1.4, Sept 2019

Functions

```
// function with no parameters or result
func printHello() {
  print( "Hello")
}
printHello()  // usage

// function with one parameter, no result
func printMessage(message: String) {
  print( message )
}
printMessage(message: "Well, hello")  // usage

// To not need to name parameter in usage,
// would need to define printMessage as follows
// func printMessage(_ message: String)

// function with two parameters, one result
func concat(s1: String, s2: String ) -> String {
   return s1 + " and " + s2
}
let billAndBen = concat(s1: "Bill", s2: "Ben")

// function with no params, tuple as result
func returnTuple() -> (String, Int) {
  return( "Page not found", 404 )
}
let result = returnTuple()
// String in result can be accessed as result.0
// Int in result can be accessed as result.1
```

Structures

```
struct Person{
  // properties
  let firstName: String
  let lastName: String

  // computed property
  var fullName: String {
    return firstName + " " + lastName
  }

  // Method - like function but on instance
   func printName(){
      print( fullName )
   }
}

//Create an instance of a Person
let newPerson = Person(firstName: "Jim",
                       lastName: "Jones")

//Call method for instance newPerson
newPerson.printName()

let oldPerson = newPerson
//When you assign a structure to a new variable,
//its value is COPIED (unlike classes).

// Structures have automatically generated
//initialisers, but you can also write your own.
```

Classes

```
//Class syntax would be identical for the Person
//class, but need to write an initialiser.
class Person{
  let firstName: String
  let lastName: String

   // Needs this initialiser
   init(fName: String, lName: String) {
      firstName = fName
      lastName = lName
   }

  var fullName: String {
      return firstName + " " + lastName
  }

   func printName(){
      print( fullName )
   }
}

let newPerson = Person(fName: "Jim",
                       lName: "Jones")

newPerson.printName()
let oldPerson = newPerson
//When you assign a class to a new variable,
//both variables point to the same structure
```

The Swift Language - Brief Overview (4)

Chris Price v1.4, Sept 2019

Class inheritance

```
// Can declared subclass of existing class
// Subclass inherits properties and methods
// Extra properties and methods can be declared,
// but extra properties must be initialised.
// Where methods are replaced by more specific
//versions, they need the qualifier 'overrides'

class Doctor: Person{
  let specialism: String
   // Needs this initialiser
   init(fName: String, lName: String, spec: String) {
      specialism = spec
      super.init(fName: fName,lName: lName)
   }

   override func printName(){
      print( "Dr " + fullName+" does "+specialism)
   }

   func seePatient( patientName: String ) {
      print( "Hullo, " + patientName
             + ", I'm Dr " + fullName )
   }
}
let doc = Doctor(fName: "Jane",
                 lName: "Jones", spec: "Pediatrics")
doc.printName()
doc.seePatient(patientName: "George")
```

Enumerations

```
enum TempType {
   case degF
   case degC
}
var tempType = TempType.degC
if tempType == .degF {
   print("In Fahrenheit")
} else {
   print("In centigrade")
}

// More complex example
enum DaysOfWeek {
   case monday, tuesday, wednesday,
          thursday, friday, saturday, sunday
}

var today = DaysOfWeek.monday
switch today{
case .saturday, .sunday:
   print( "Chill - it's the weekend" )
default:
   print( "Apply nose to grindstone" )
}
// Will print "Apply nose to grindstone"

today = .saturday
// We know type of today now,
// so don't need to say "DaysOfWeek"
```

Dictionaries

```
//Create a dictionary containing our pets
var pets = ["Dave": "Fish", "Chaz": "Dog"]
pets["Idris"] = "Pig"  //Adds Idris to pets
pets.isEmpty      // returns false
pets.count        // returns 3
pets["Idris"]     // returns " Pig"
pets["Dog"]       // returns nil

// Update value for Chaz
pets["Chaz"] = "Bad Dog"

// Loop over key/value pairs
for (name, animal) in pets {
    print( "\(name) is a \(animal)" )
}

//Delete Chaz from dictionary
pets["Chaz"] = nil // Deletes Chaz entry

// Loop over keys
for name in pets.keys {
   print( "\(name) is a pet" )
}

// Loop over values
for animal in pets.values {
   print( "We have a \(animal)" )
}
```

Session 2: Introduction to using Xcode

- Making first app / toolkit features
- MVC and linking code to Storyboards
- Using the documentation
- Introduction to UIKit
- *Lab 2: Building and running single screen apps on simulator and device*

- This will cover lessons 1.5 to 1.8 of Development in Swift Fundamentals

How do you make a multi-screen app?

Case Study: Crossflow Energy

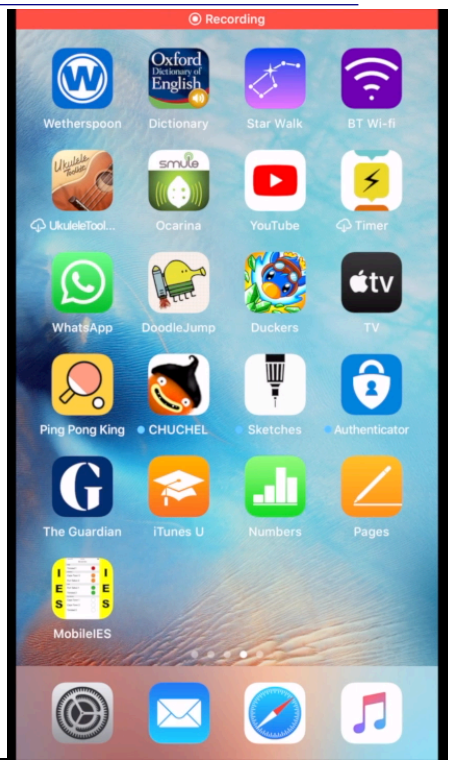
At the core of Crossflow's MODULAR MICROGENERATION SYTEMS are Crossflow's SMART Wind Turbines integrated with market leading solar and battery technologies.

Turbine
Crossflow SMART TURBINE

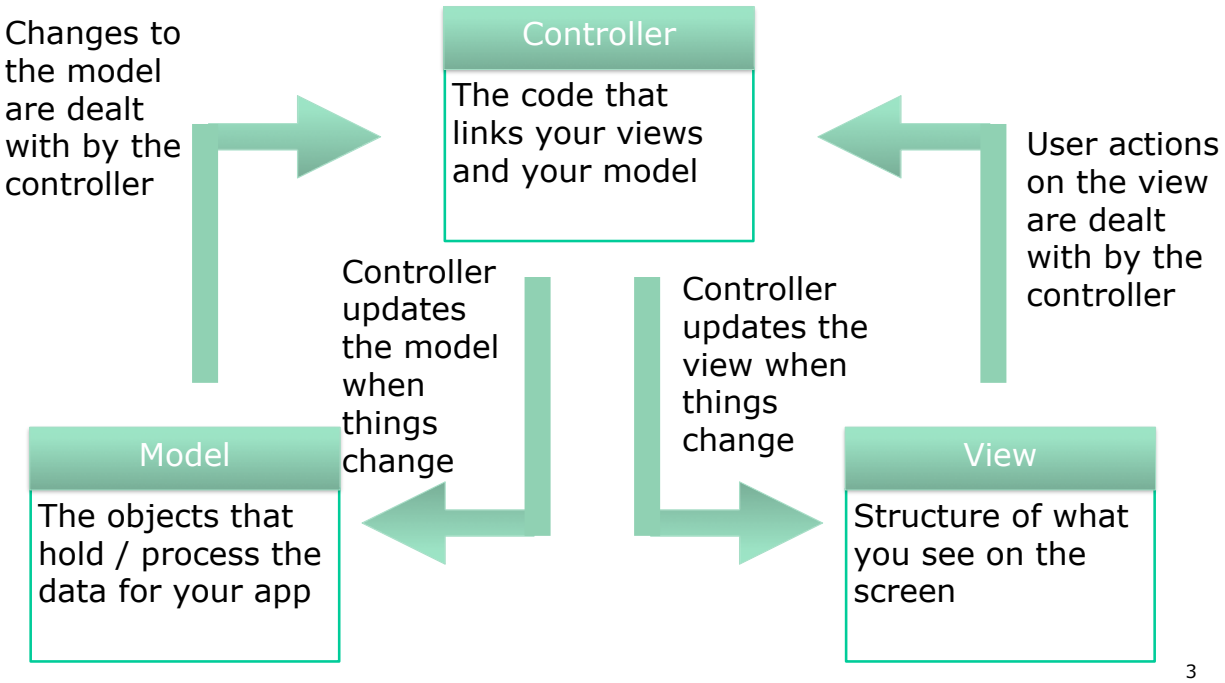
PV Array
Flexi or Fixed Solar pV with ground and tower mounting options

Battery
Latest rack mounted battery technology providing consistent power supply

Generation or Grid
Other Energy Generation inputs optional

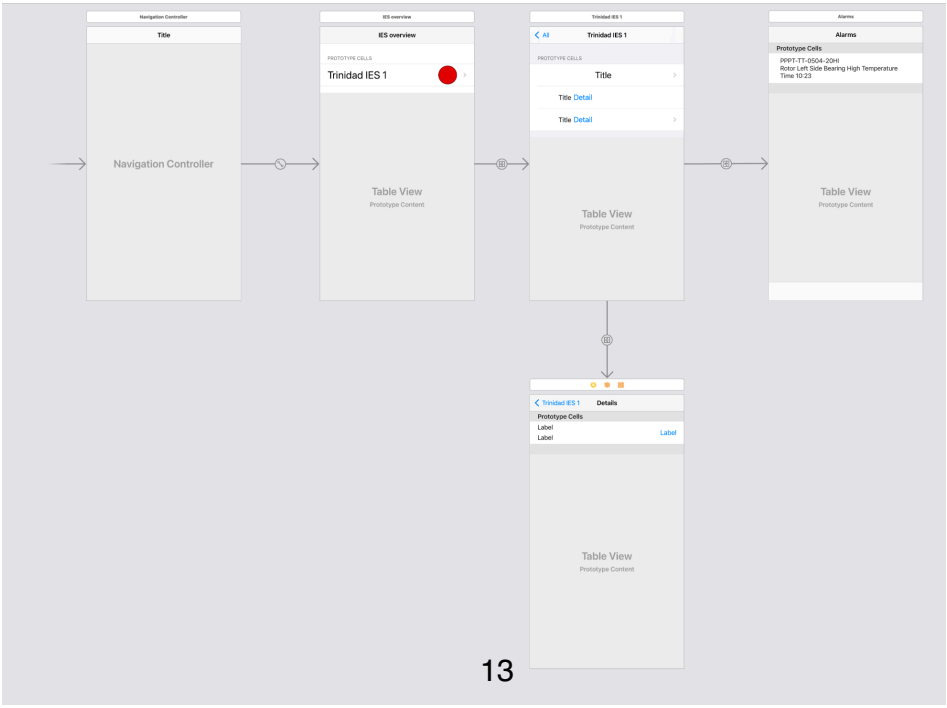


Model-View-Controller in Xcode

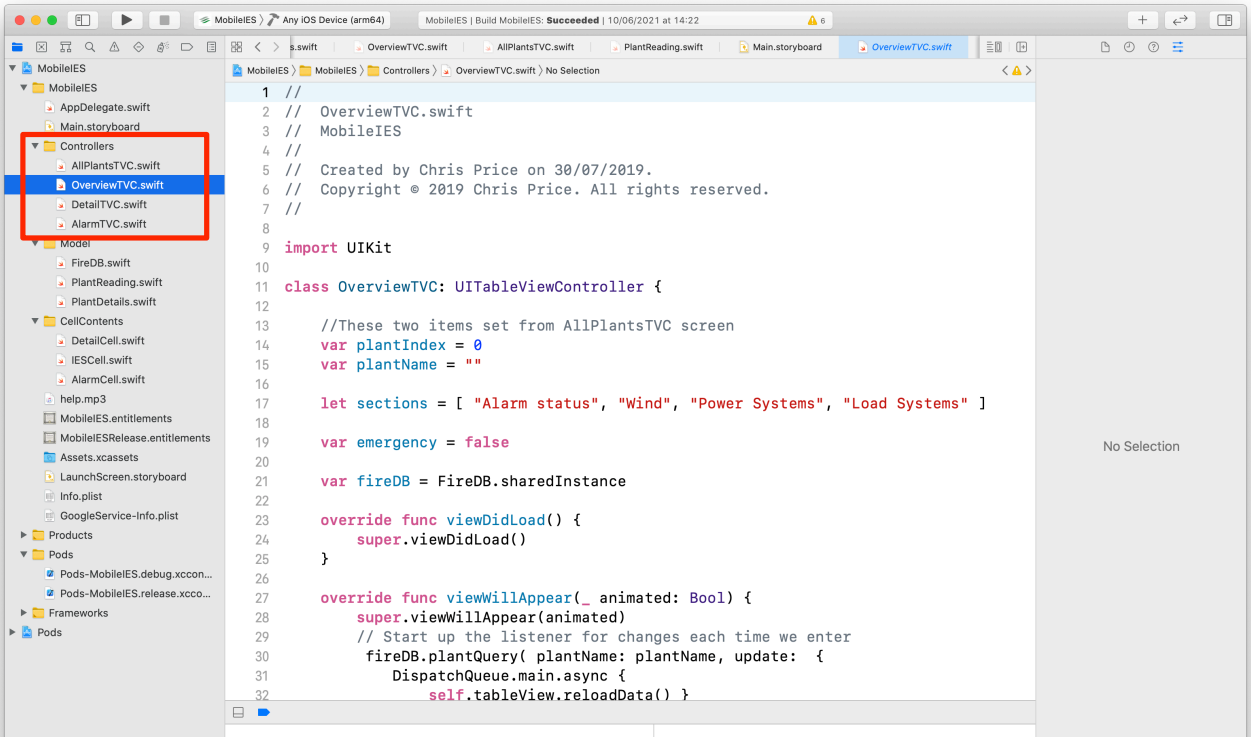


Views are made in Interface Builder

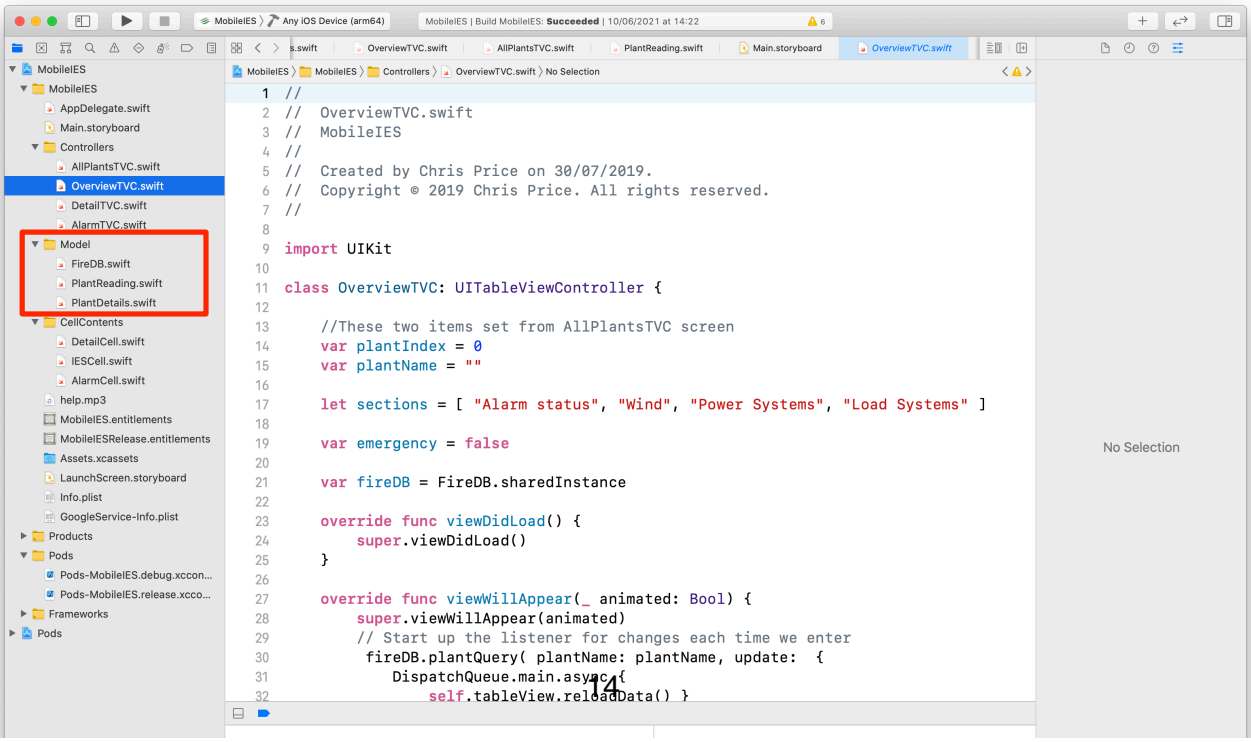
in simple case, one view for each screen



There will be view controller code for each of those screens



Model code will be a set of classes for holding and operating on data objects



Session 3: UIKit: Views and controls

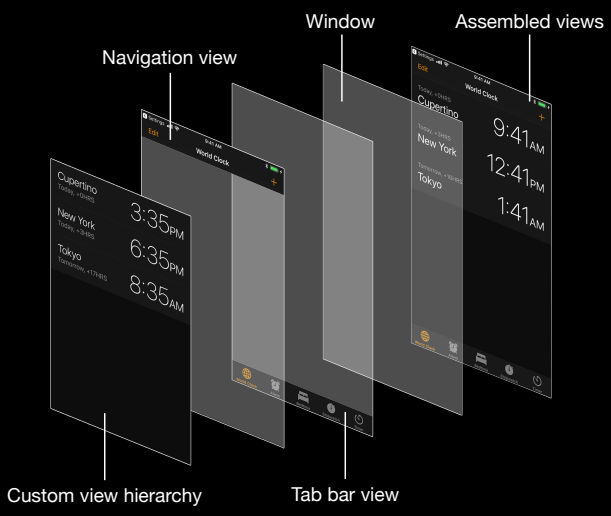
- Attributes of common views
- Adding controls to your app
- Adding more views
- *Lab 3: Building apps with more views and controls*

- This will cover lessons 2.7 to 2.9 of Development in Swift Fundamentals

1

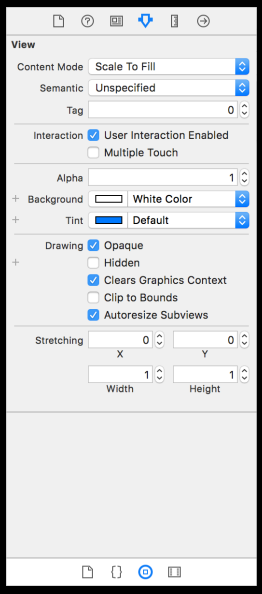
Unit 2—Lesson 7: Introduction to UIKit (p227 in book)

Common system views

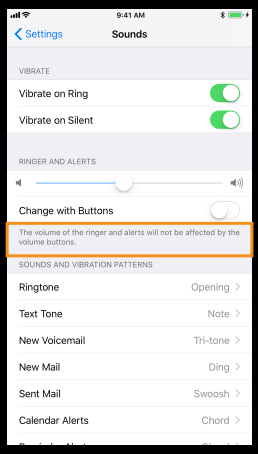


Common system views
Configuration

Property	Description
Size	Width and height of the view
Position	Position of the view onscreen
Alpha	Transparency of the view
Background Color	Background color that will be displayed
Tag	Integer that you can use to identify view objects



Label (UILabel)



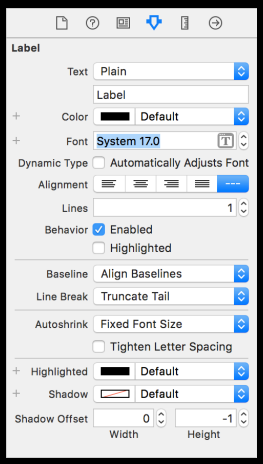
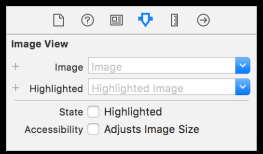
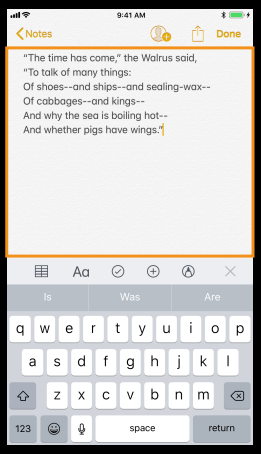
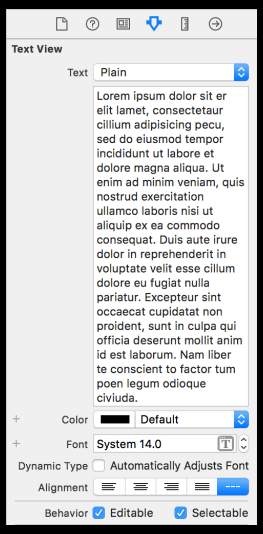


Image view (UIImageView)

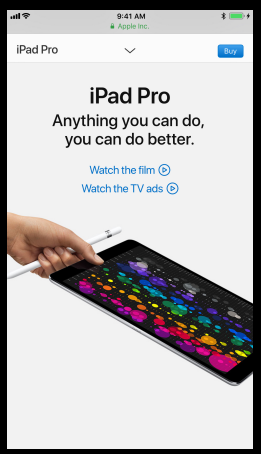


Text view (UITextView)





Scroll view (UIScrollView)



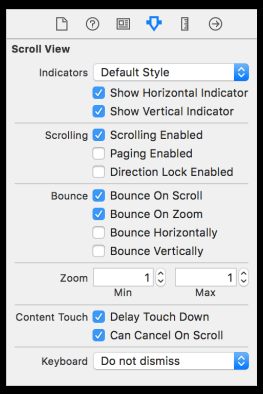
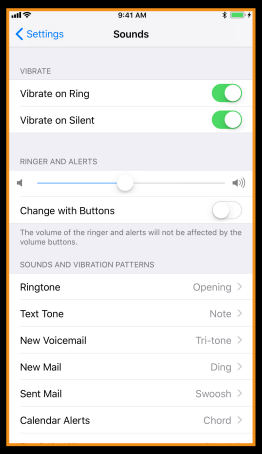
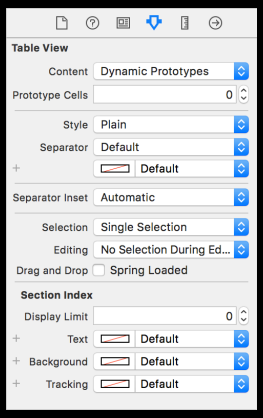
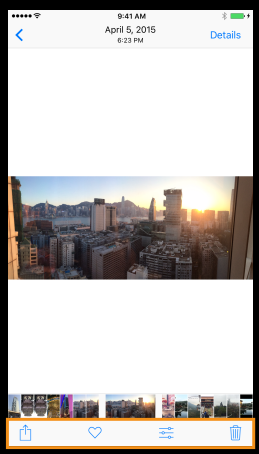


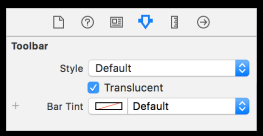
Table view (UITableView)



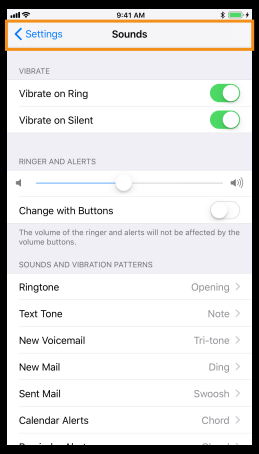


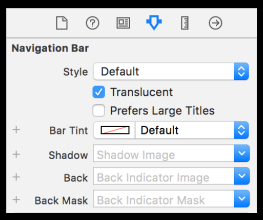
Toolbars (UIToolbar)



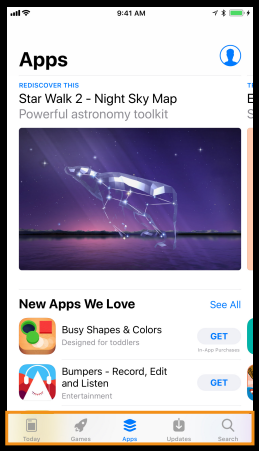


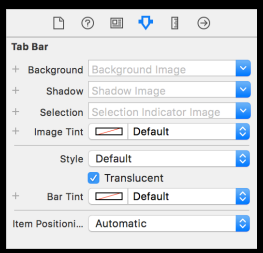
Navigation bars (UINavigationBar)





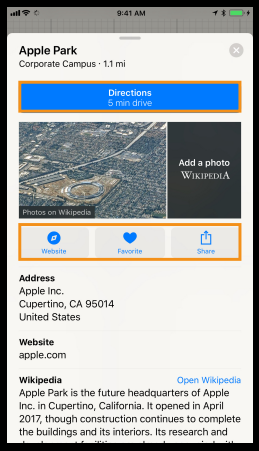
Tab bars (UITabBarController)

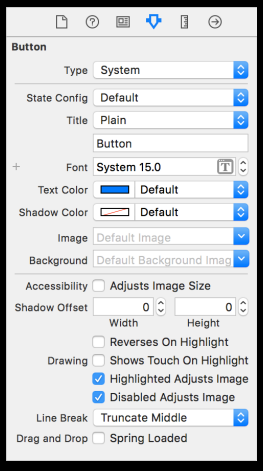




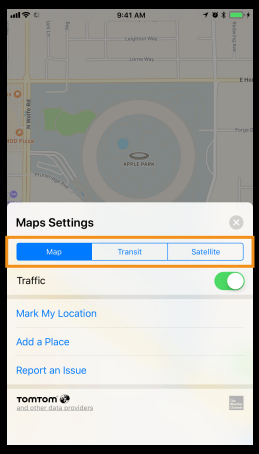
Controls

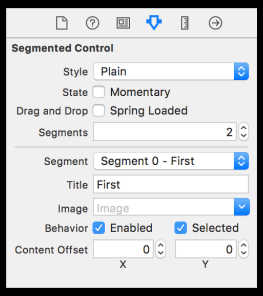
Buttons (UIButton)



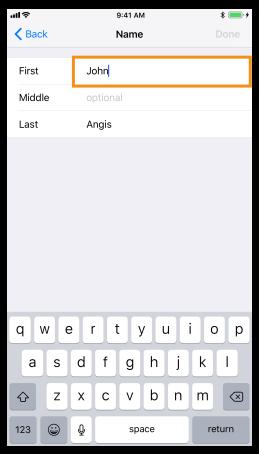


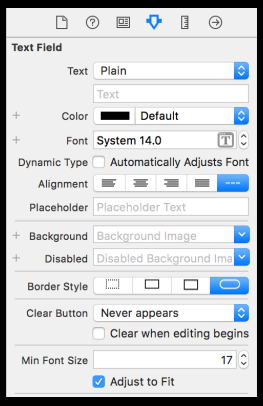
Segmented controls (UISegmentedControl)



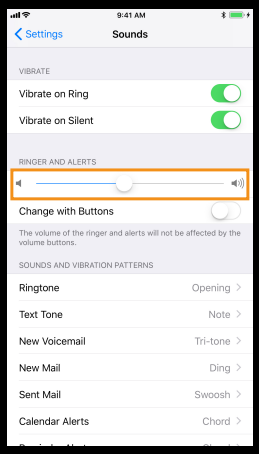


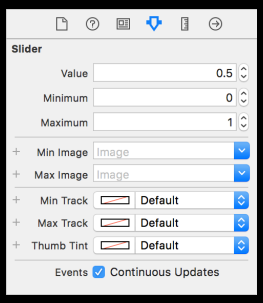
Text fields (UITextField)



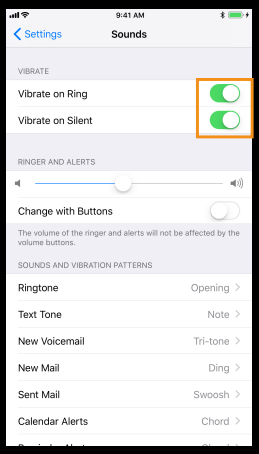


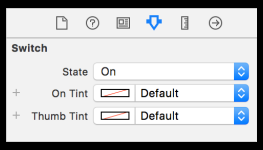
Sliders (UISlider)



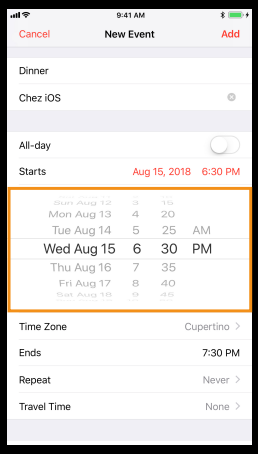


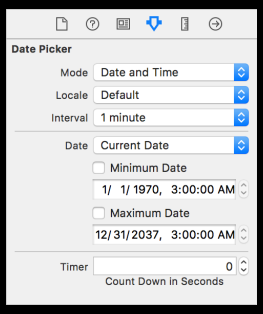
Switches (UISwitch)



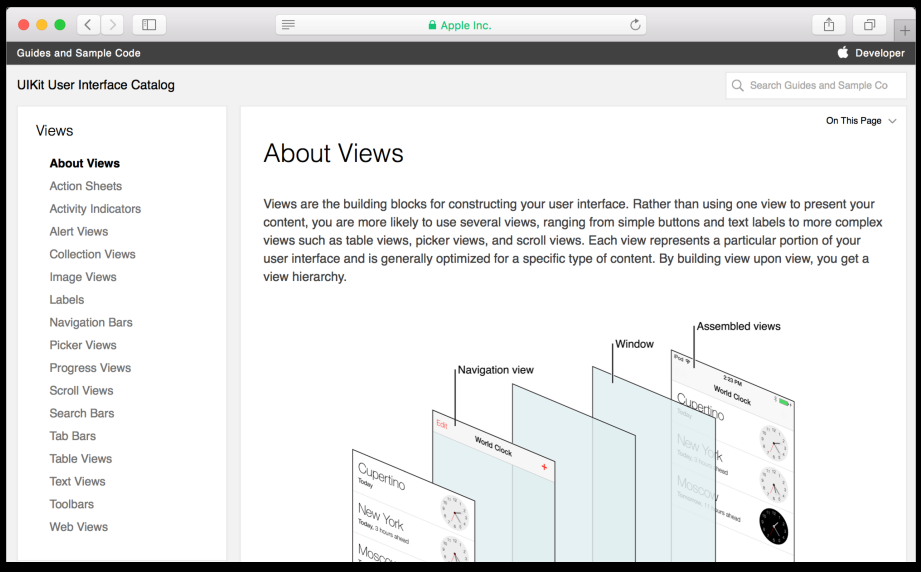


Date pickers (UIDatePicker)





UIKit User Interface Catalog



Practical

We will work through exercises in 2-08 Displaying Data (pp253-260)

Then do 2-09 Controls in Action (page 265 to top of page 283)

Session 4: Functions

- Attributes of common views
- Adding controls to your app
- Adding more views
- *Lab 3: Building apps with more views and controls*

- This covers lesson 2.2 of Development in Swift Fundamentals

1

Unit 2—Lesson 2: Functions

Functions

```
tieMyShoes()
```

```
makeBreakfast(food: "scrambled eggs", drink: "orange juice")
```

Functions
Defining a function

```
func functionName (parameters) -> ReturnType {
    // Body of the function
}
```

```
func displayPi() {
  print("3.1415926535")
}

displayPi()
```

```
3.1415926535
```

Parameters

```
func triple(value: Int) {
  let result = value * 3
  print("If you multiply \(value) by 3, you'll get \(result).")
}

triple(value: 10)
```

```
If you multiply 10 by 3, you'll get 30.
```

Parameters
Multiple parameters

```
func multiply(firstNumber: Int, secondNumber: Int) {
  let result = firstNumber * secondNumber
  print("The result is \(result).")
}

multiply(firstNumber: 10, secondNumber: 5)
```

```
The result is 50.
```

Return values

```
func multiply(firstNumber: Int, secondNumber: Int) -> Int {
  let result = firstNumber * secondNumber
  return result
}
```

Return values

```
func multiply(firstNumber: Int, secondNumber: Int) -> Int {
 return firstNumber * secondNumber
}
```

```
let myResult = multiply(firstNumber: 10, secondNumber: 5)
print("10 * 5 is \(myResult)")
```

```
print("10 * 5 is \(multiply(firstNumber: 10, secondNumber: 5))")
```

Argument labels

```
func sayHello(firstName: String) {
  print("Hello, \(firstName)!")
}

sayHello(firstName: "Amy")
```

Argument labels

```
func sayHello(to: String, and: String) {
  print("Hello \(to) and \(and)")
}

sayHello(to: "Luke", and: "Dave")
```

Argument labels
External names

```
func sayHello(to person: String, and anotherPerson: String) {
  print("Hello \(person) and \(anotherPerson)")
}

sayHello(to: "Luke", and: "Dave")
```

Argument labels
Omitting labels

```
print("Hello, world!")
```

```
func add(_ firstNumber: Int, to secondNumber: Int) -> Int {
    return firstNumber + secondNumber
}

let total = add(14, to: 6)
```

Default parameter values

```
func display(teamName: String, score: Int = 0) {
  print("\(teamName): \(score)")
}

display(teamName: "Wombats", score: 100)
display(teamName: "Wombats")
```

```
Wombats: 100
Wombats: 0
```

Unit 2—Lesson 2
Lab: Functions

There is a short Playground lab (called “Quick functions.playground”) in

Try that.

If you have problems with that, work through chapter 2.2 from the book, and try the function exercises in the Longer Functions playground in your own time.

Session 5: Prototyping an app interface

- Deciding what goes into the app
- Making the screens for a multi-screen app and joining them
- Trying out the app on potential users
- Taking Apple design considerations and HCI guidelines into account
- *Lab 5: Building an app prototype without code*

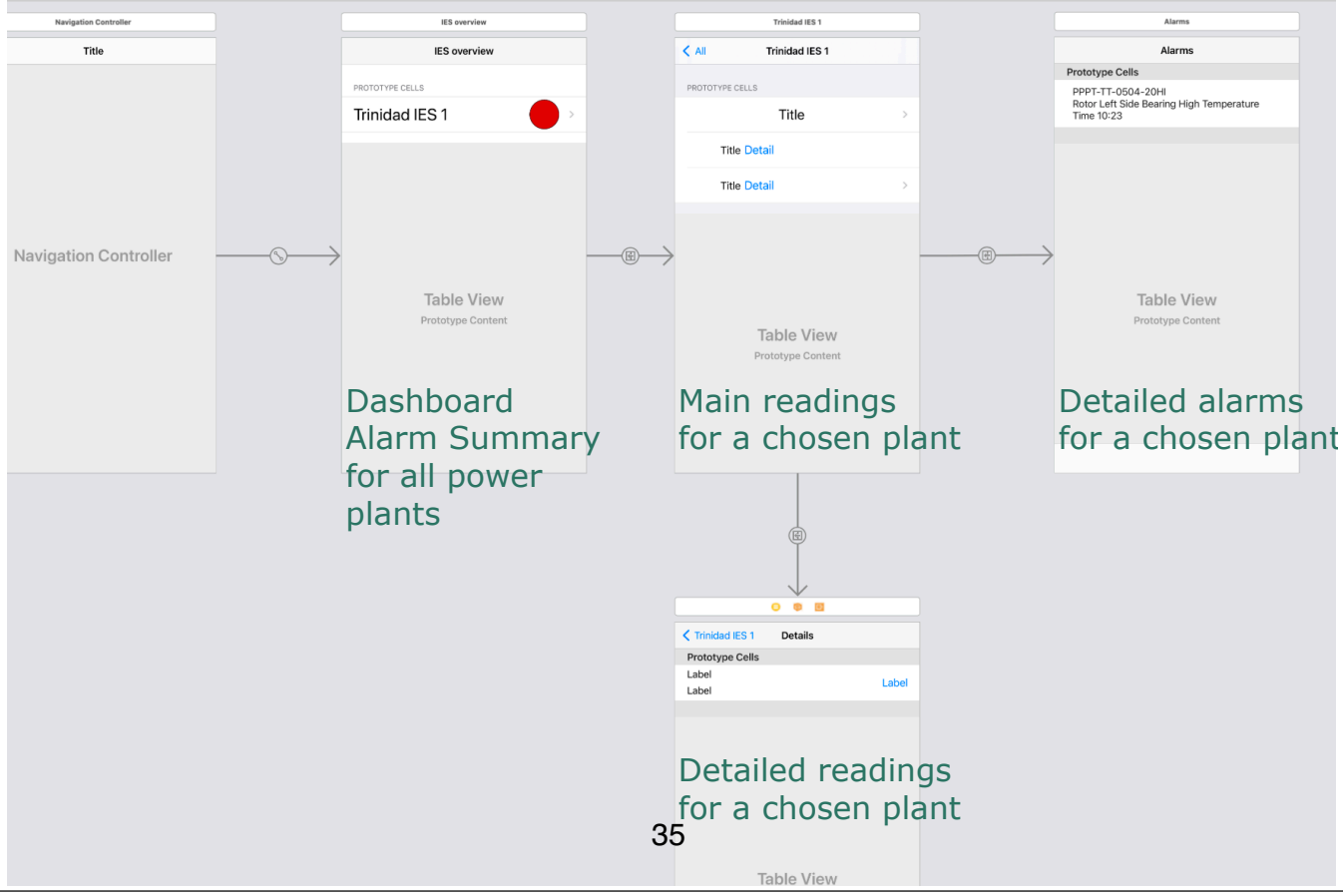
- This material not well covered in Apple coding books

Party Talk

How do we turn an idea into an app

- Prototyping
- Xcode and Interface Builder can do this very efficiently and very effectively
- Build the views and add realistic data
- When we agree on the screens that will make up the app, we have made several steps towards having a real app
- Lets look at the Crossflow example again

4 types of screen in app



4 types of screen in app

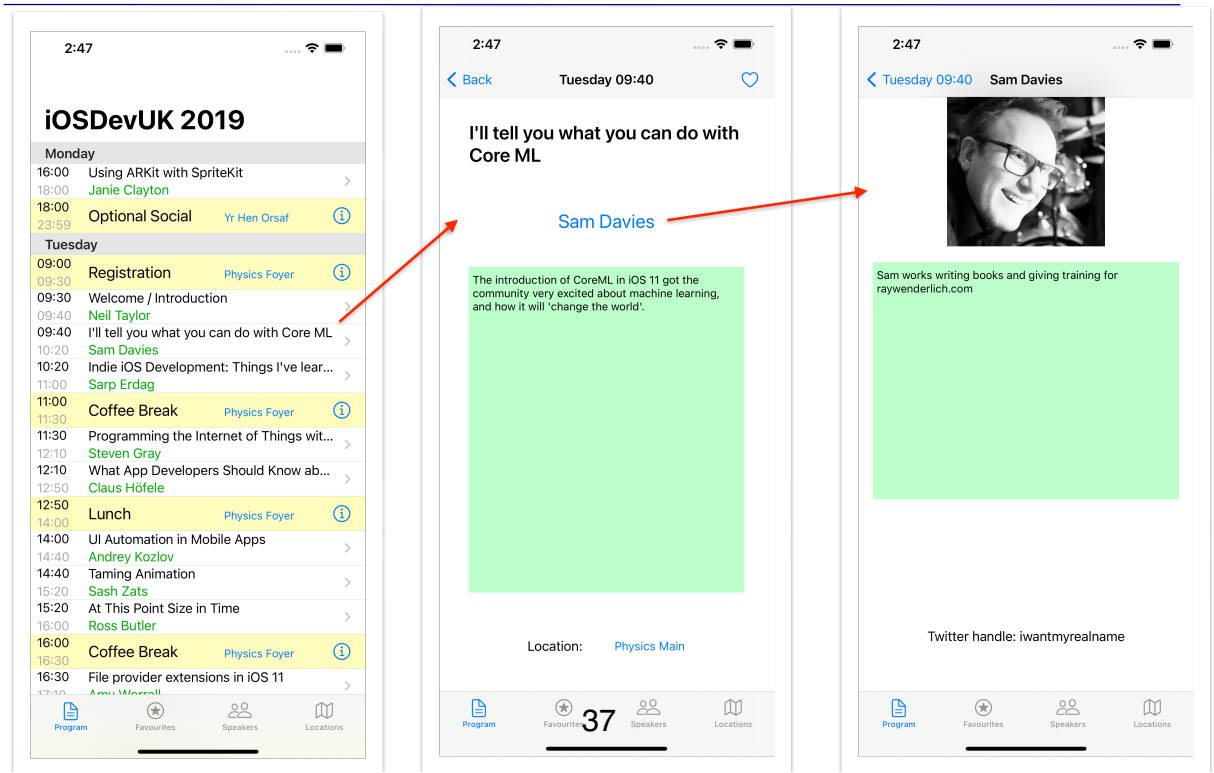
Having done this prototype

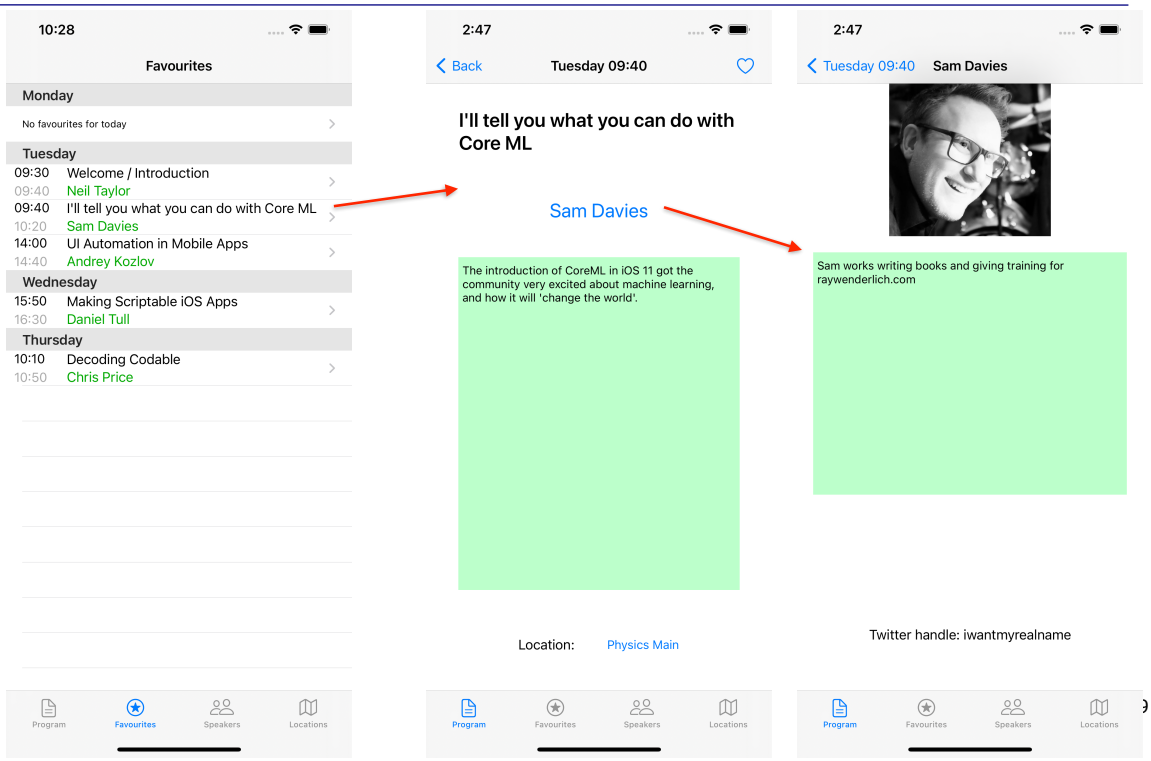
- Users can try out the prototype
- They can interact with the prototype as if it was the real app (but with static data)
- We can assess whether they have all the info they need in specific situations
- We can produce screenshots and use them in a design document
- We can make view controllers to fill out the app with its actual behaviour

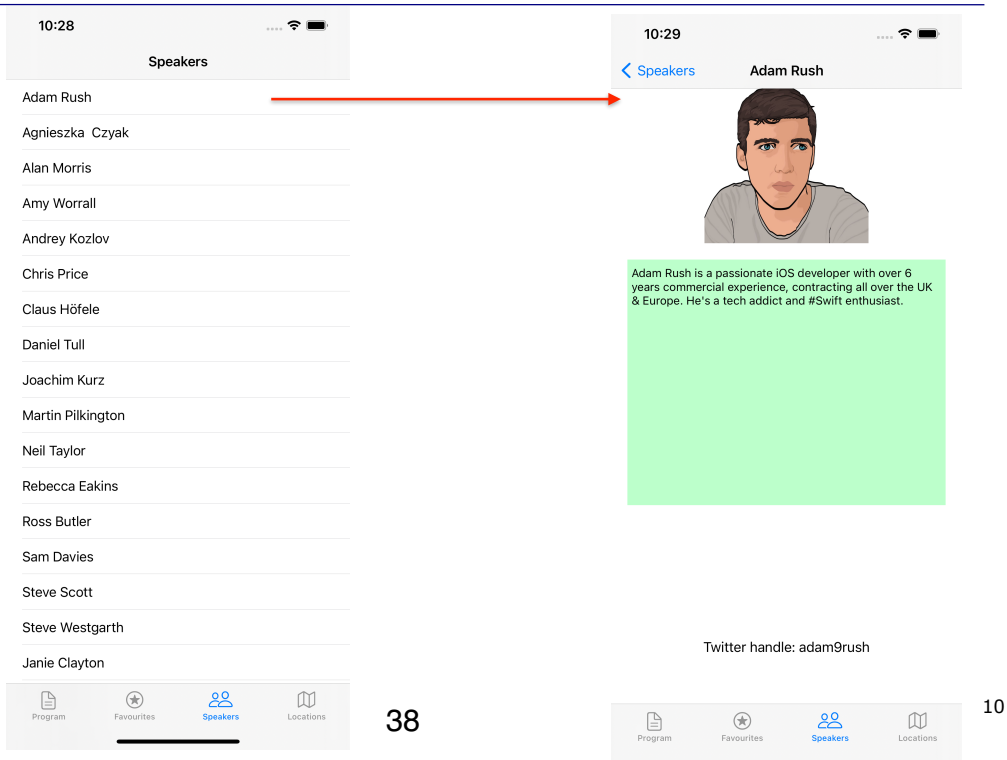
Conference Attendee App
A case study through the course

- I run the UK's largest iOS development conference each year (iosdevuk.com)
- We make an app each year so that the attendees can see:
  - the schedule of the conference
  - the details of talks
  - The details of speakers
  - Where talks, dinners etc are being held
  - And can bookmark what sessions they want to attend

We are ready to make a non-functional prototype

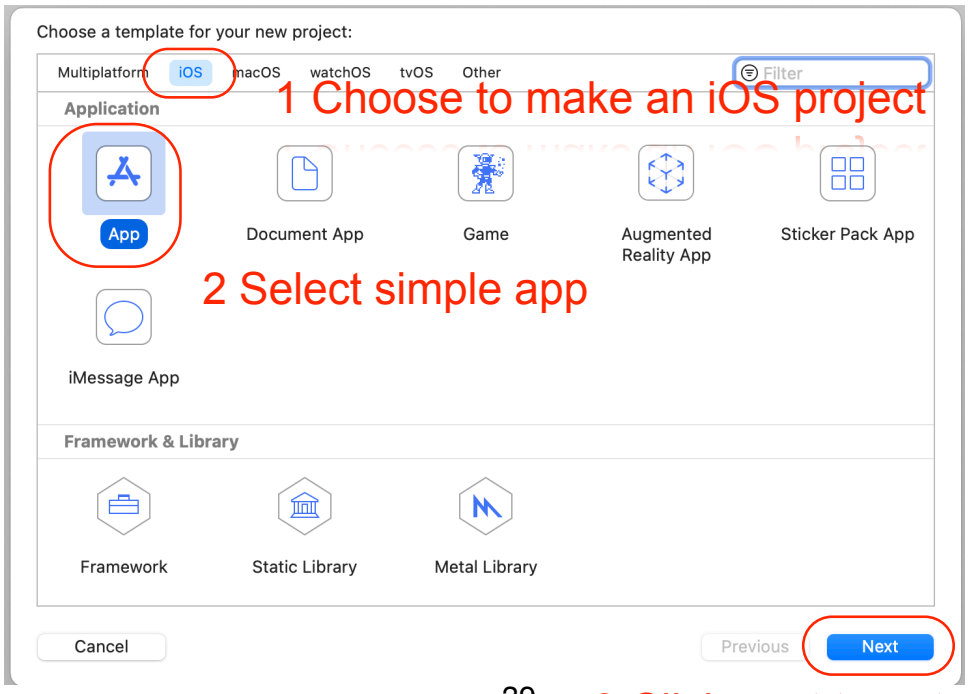



10:28
Favourites
Monday
No favourites for today
Tuesday
09:30 Welcome / Introduction
09:40 Neil Taylor
09:40 I'll tell you what you can do with Core ML
10:20 Sam Davies
14:00 UI Automation in Mobile Apps
14:40 Andrey Kozlov
Wednesday
15:50 Making Scriptable iOS Apps
16:30 Daniel Tull
Thursday
10:10 Decoding Codable
10:50 Chris Price
Program
Favourites
Speakers
Locations
2:47
Back
Tuesday 09:40
I'll tell you what you can do with Core ML
Sam Davies
The introduction of CoreML in iOS 11 got the community very excited about machine learning, and how it will 'change the world'.
Location:
Physics Main
2:47
Tuesday 09:40
Sam Davies
Sam works writing books and giving training for raywenderlich.com
Twitter handle: iwantmyrealname


10:28
Speakers
Adam Rush
Agnieszka Czyak
Alan Morris
Amy Worrall
Andrey Kozlov
Chris Price
Claus Höfele
Daniel Tull
Joachim Kurz
Martin Pilkington
Neil Taylor
Rebecca Eakins
Ross Butler
Sam Davies
Steve Scott
Steve Westgarth
Janie Clayton
Program
Favourites
Speakers
Locations
10:29
Speakers
Adam Rush
Adam Rush is a passionate iOS developer with over 6 years commercial experience, contracting all over the UK & Europe. He's a tech addict and #Swift enthusiast.
Twitter handle: adam9rush

Making prototype app - Step 1

- In Xcode, choose File/New/Project
- Following screen will show:



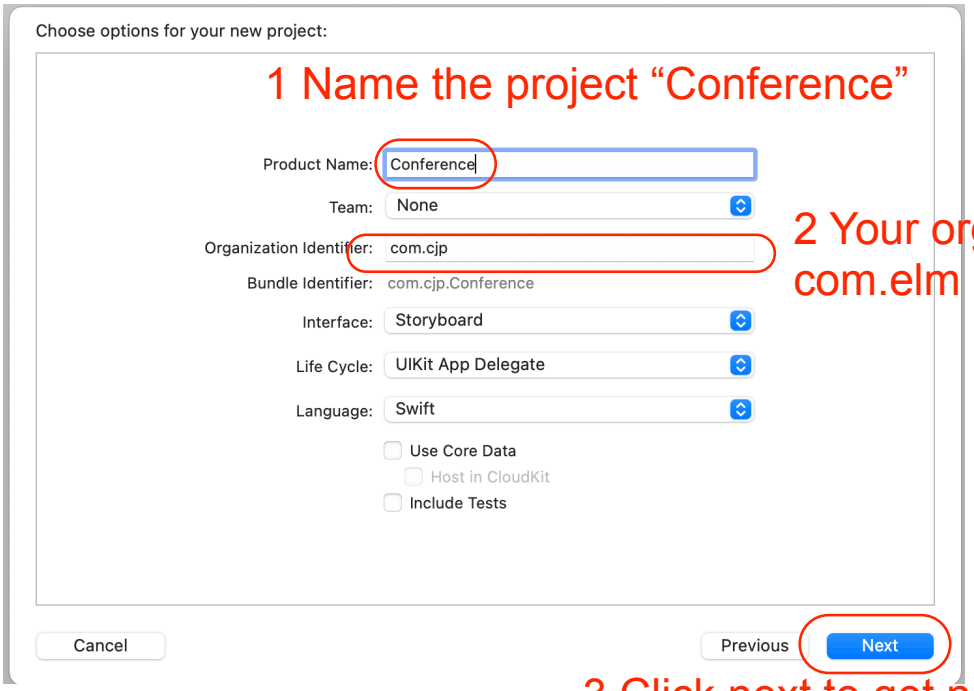
1 Choose to make an iOS project

2 Select simple app

3 Click next to get next slide

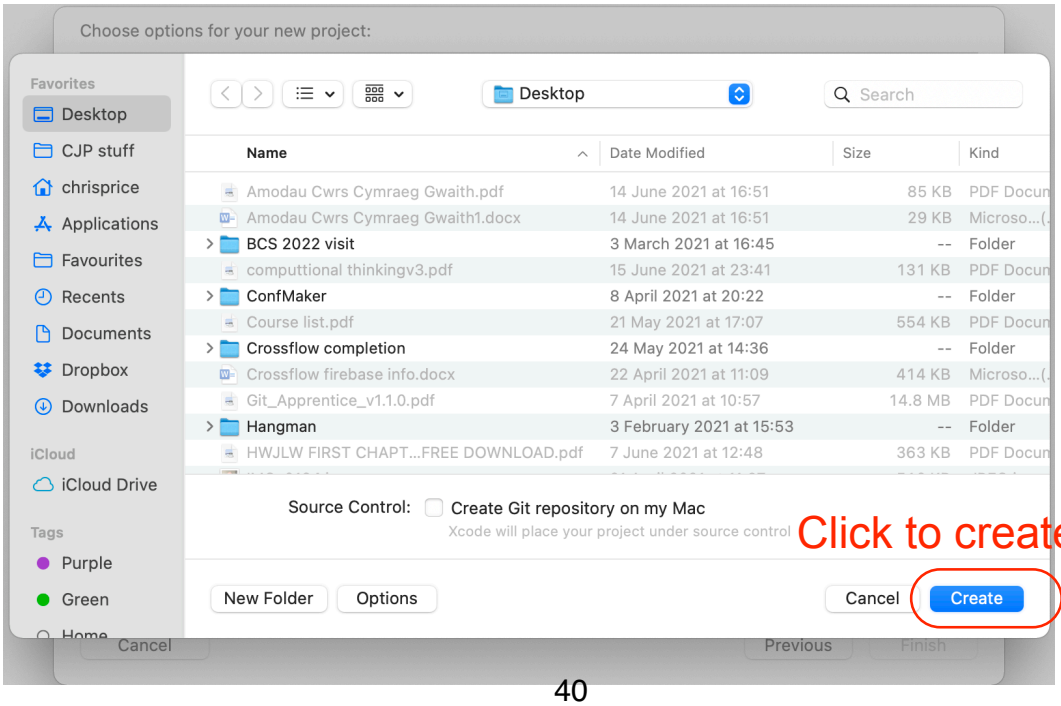
Step 2

- On this screen we fill in project details
- You need to check all details are right



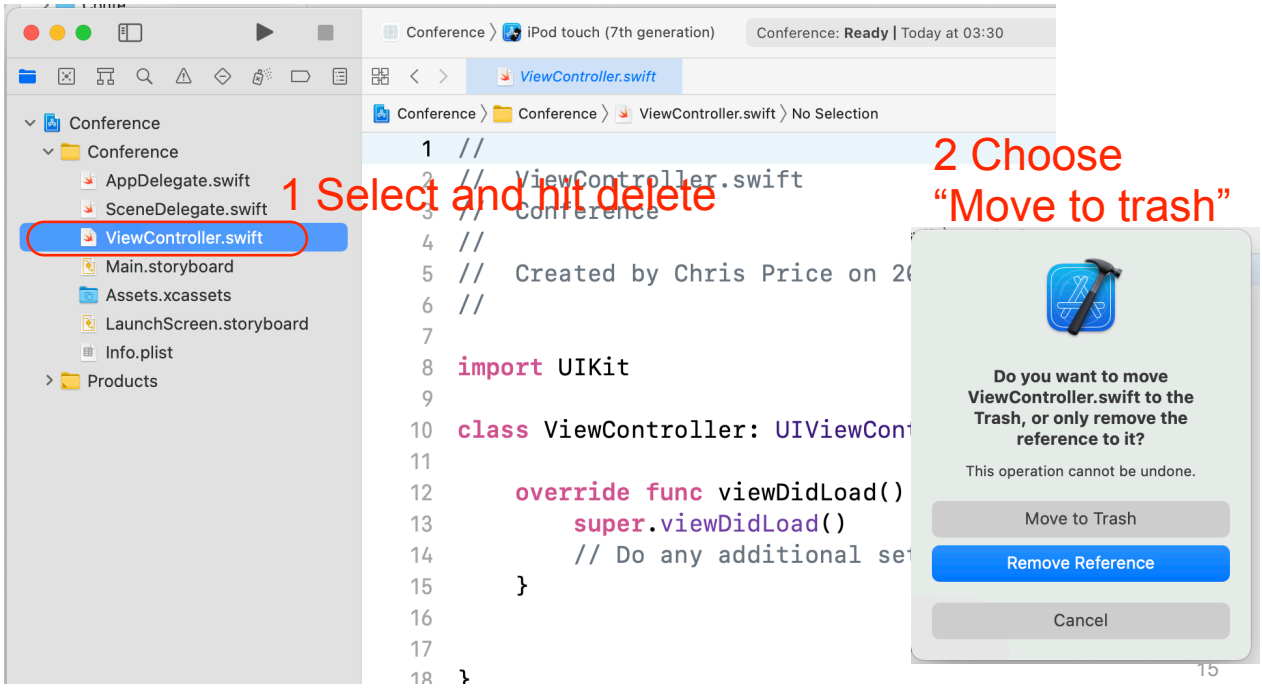
Step 3

- Final setup screen - we decide where we want to save the project, and whether we are using Git



Step 4

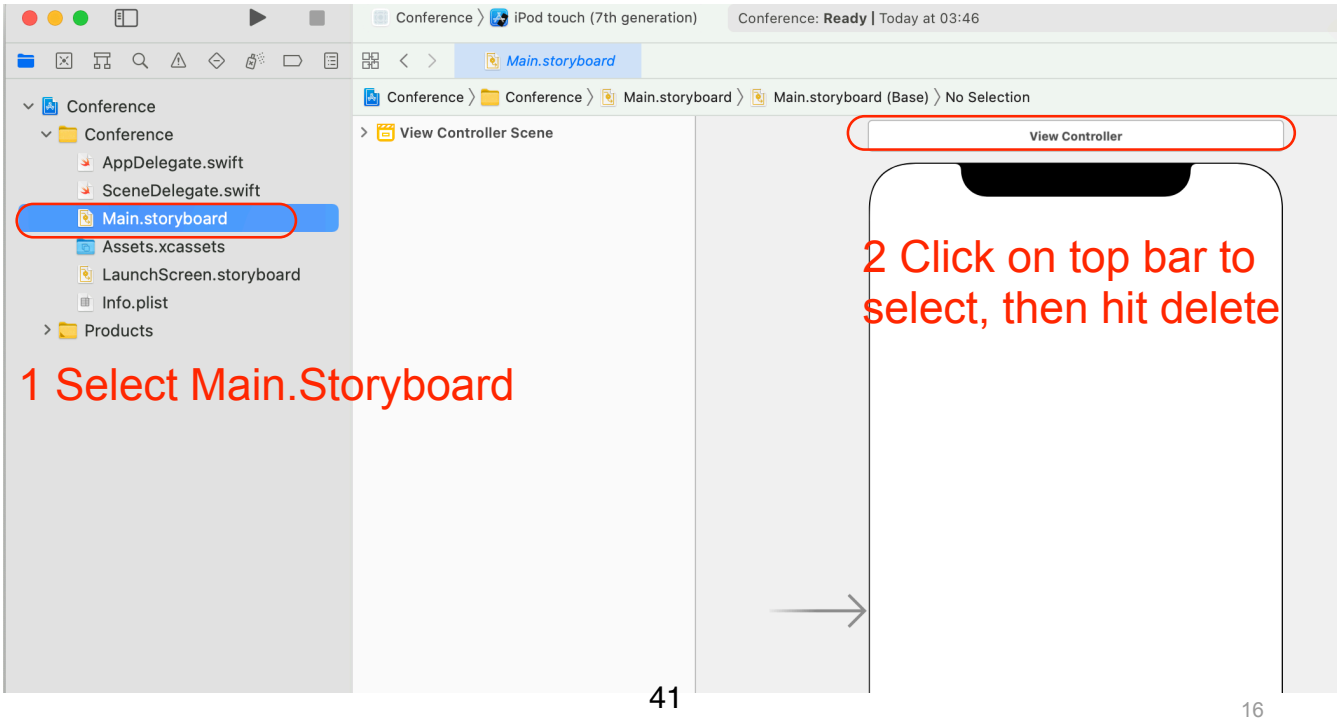
Delete ViewController.swift as we won't be using it.
•Select the file and hit delete it - choose “Move to Trash” at the prompt



15

Step 5

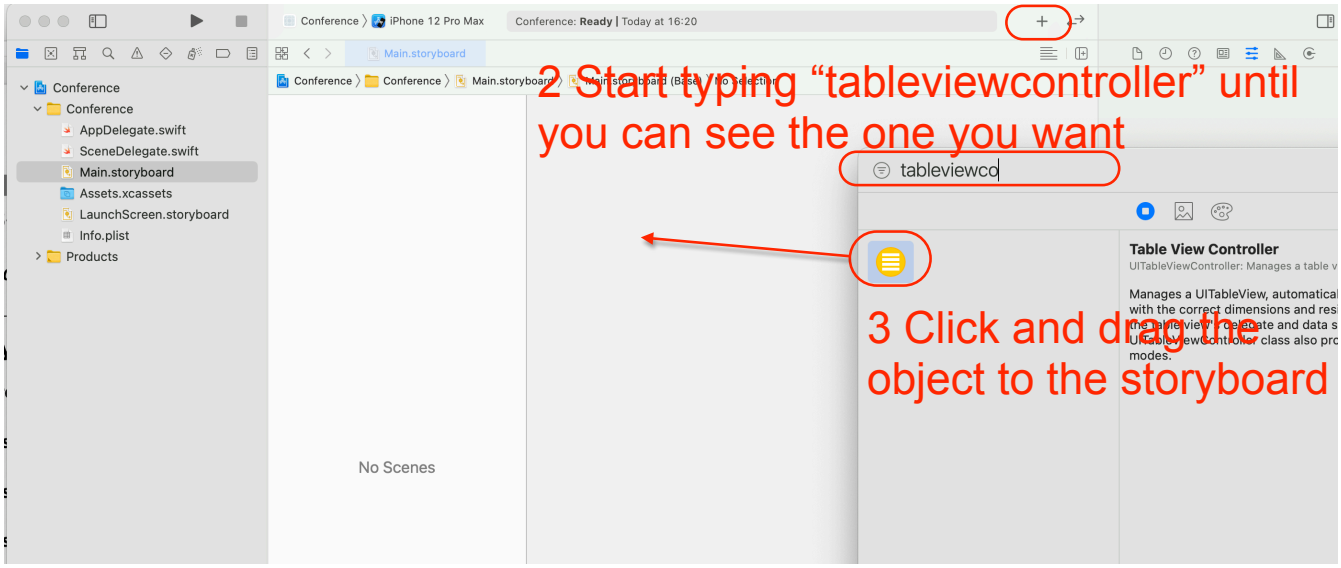
- Select Main.storyboard, then select the view controller shown in the storyboard, and hit delete



16

Step 6

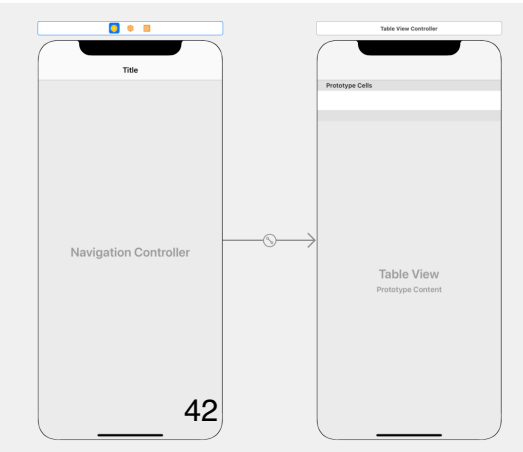
- Add a table view controller to the storyboard



Step 7

Embed the table view controller in a navigation controller

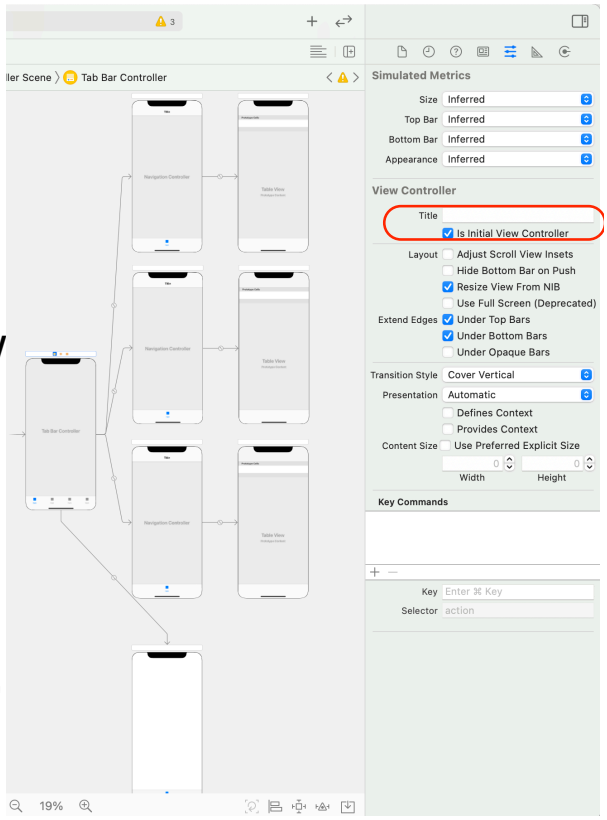
- Select the table view controller (by clicking the bar at the top in the storyboard)
- Choose menu Editor/Embed in/NavigationController
- It will then look like the screen below
- Make another two table controllers and embed each of them in a navigation controller



Step 8

Add another view controller, and make them tabs

- Add another view controller from the “+” object library
- Select all three navigation controllers and the simple view controller (click on storyboard and drag over them)
- Choose menu Editor/Embed in/TabController
- Select the tab controller and in the attribute editor on the right, select *Is InitialViewcontroller*
- This will make the tab appear when the app starts up

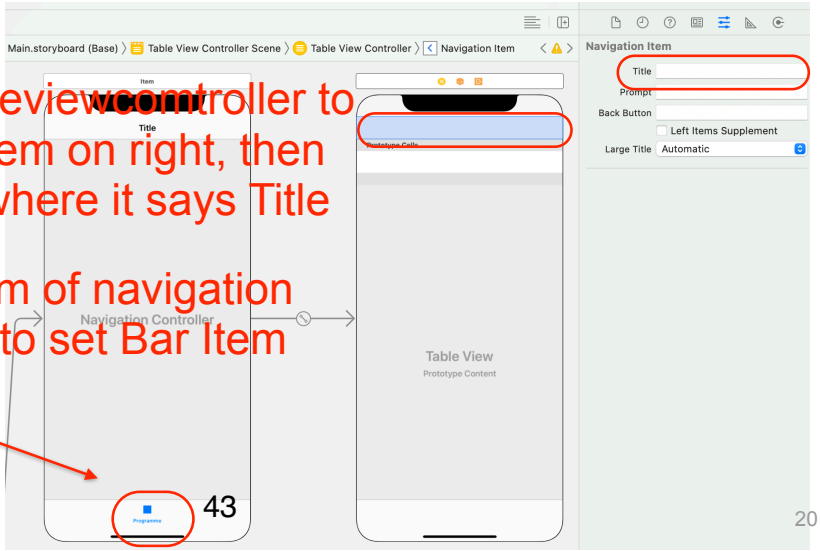


Step 9

- Click at the top of the first table view controller
- Insert the title “Programme”
- Click at the bottom of the associated navigation bar
- Type “Programme” there for the bar item. It has an associated image - select “doc.text”

1 Click at top of tableviewcontroller to get the navigation item on right, then type “Programme” where it says Title

2 Then click at bottom of navigation controller to be able to set Bar Item values

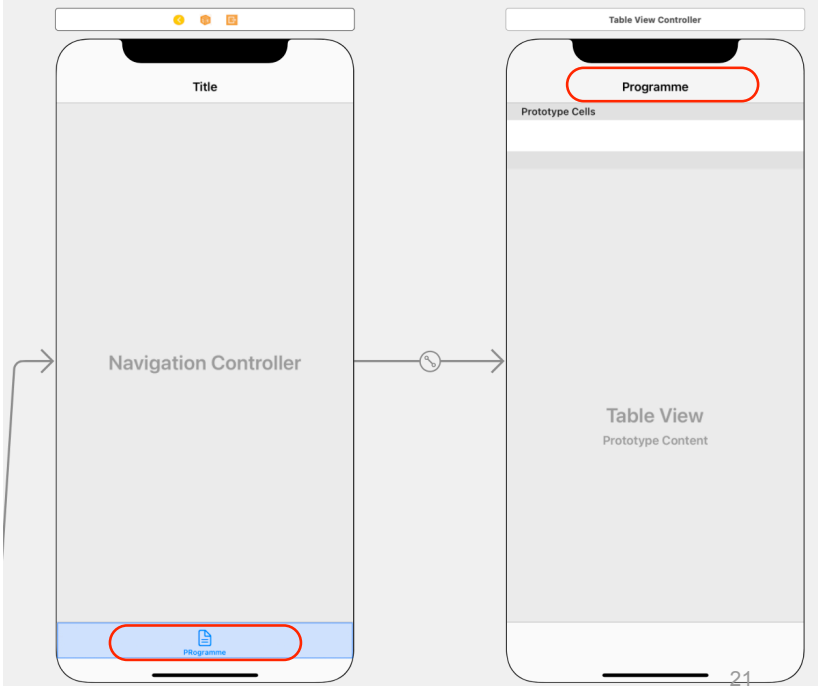


Step 10

- After step 9, the first navigation pair looks as shown

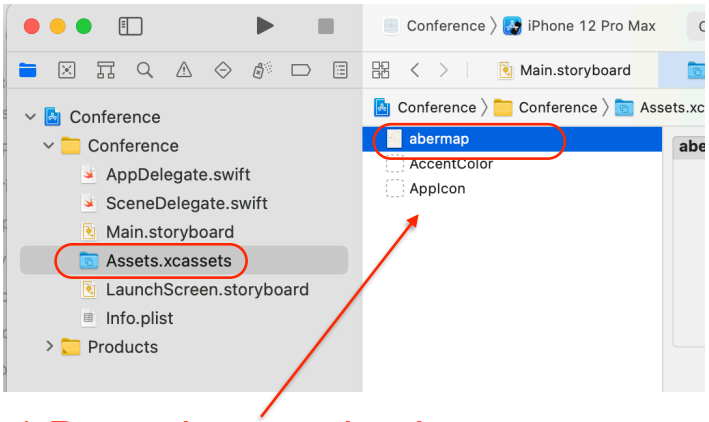
- Repeat for 2nd pair with Favourites and image star.circle

- Same for 3rd Table view controller with Speakers and image person.2

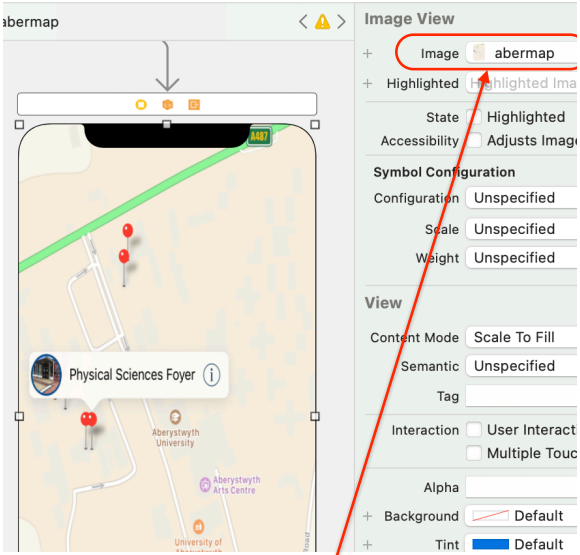


Step 11

- For the simple view controller, add the abermap.jpg provided in Session 5 to Assets, then add a image view from the object library to the view controller, and initialise it with the jpg



1 Drag abermap.jpg here



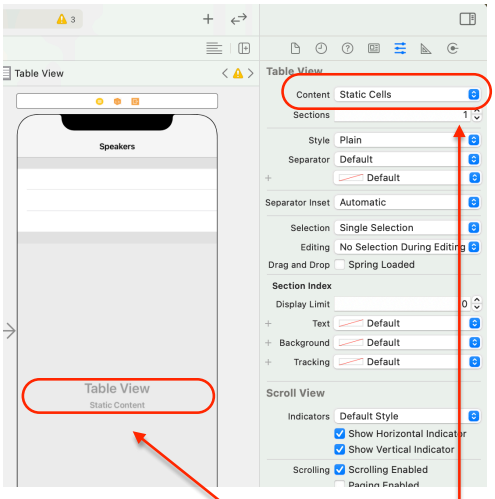
2 Having added image view, set it to abermap

Where are we?

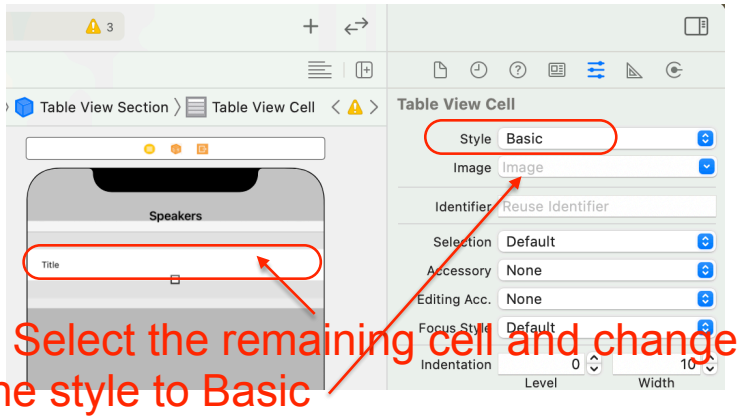
- If you run your app, it has four tabs - when you select each of the first three, you get an empty screen except for the header, and the fourth shows a map
- Next we will populate the Speaker tab with a list of Speakers, and link it to the details of a single speaker

Step 12

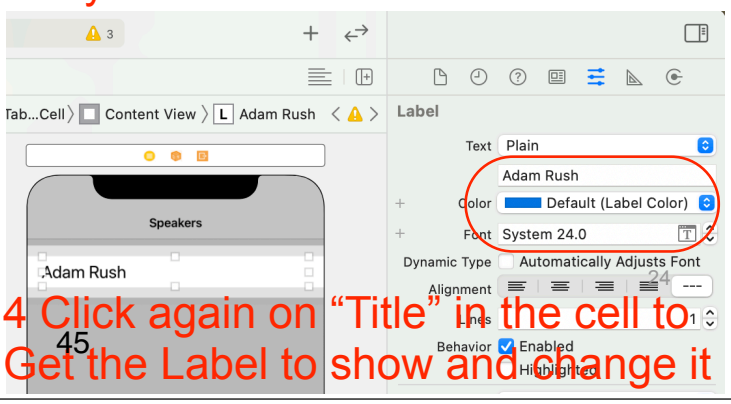
- We are going to add some static cells to the Speaker tableview controller



1 Select the Table View and set the Content to static cells
2 Click on two of the three cells created and hit delete



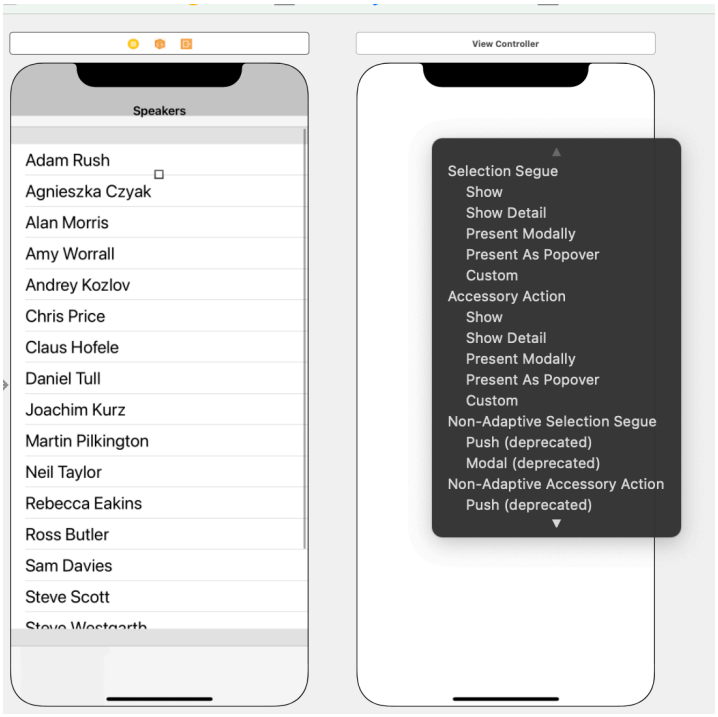
3 Select the remaining cell and change the style to Basic



4 Click again on "Title" in the cell to Get the Label to show and change it

Step 13

- Click off the table cell, then you can select it again
- You can then type command-d to duplicate the cell into a whole screen of cells, and put different names in
- Add another view controller to the right of the Speakers screen
- Click and drag from the Adam Rush cell to the new screen and choose Show on the pop up menu
- Click on the title space at the top of the new screen and add Adam Rush as a title



Step 14

- Add an image view, a text view and a label to the new screen
- Add the Adam Rush jpg to Assets, then select it to be shown on the image view
- Put some text about him in the text view
- Put his twitter handle in the label

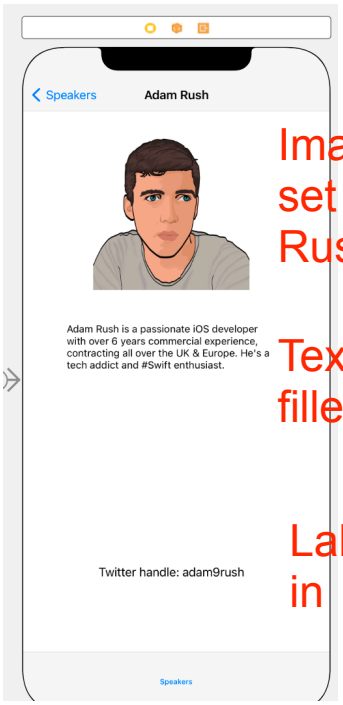


Image view, set to "Adam Rush.jpg"

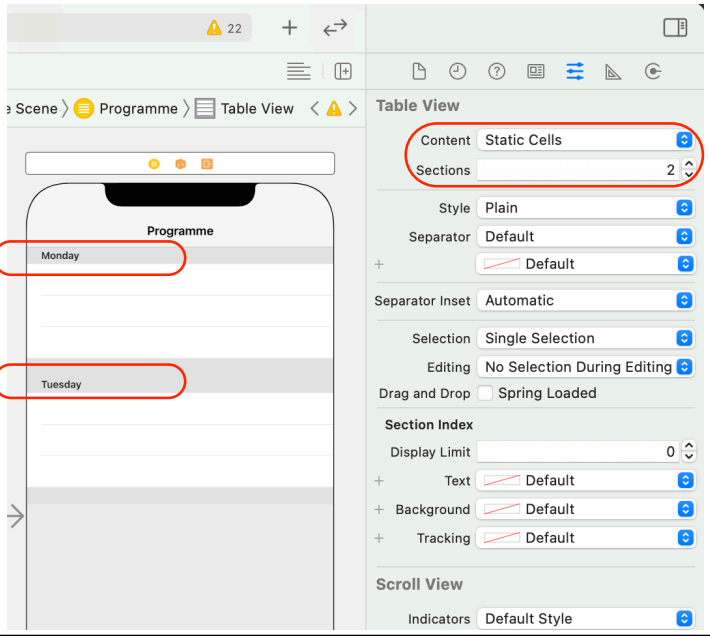
Text view, filled in

Label, filled in

- We now have the third and fourth tab choices looking good - next we will fill in the Programme

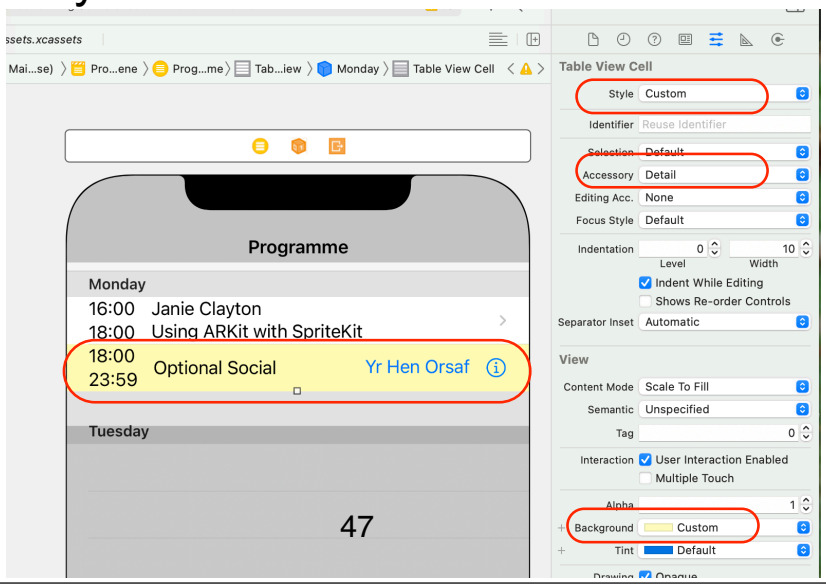
Step 15

- Make the Programme table view into static cells (as we did for Speakers in step 12)
- Change number of sections to 2
- Make section headers say “Monday” and “Tuesday”



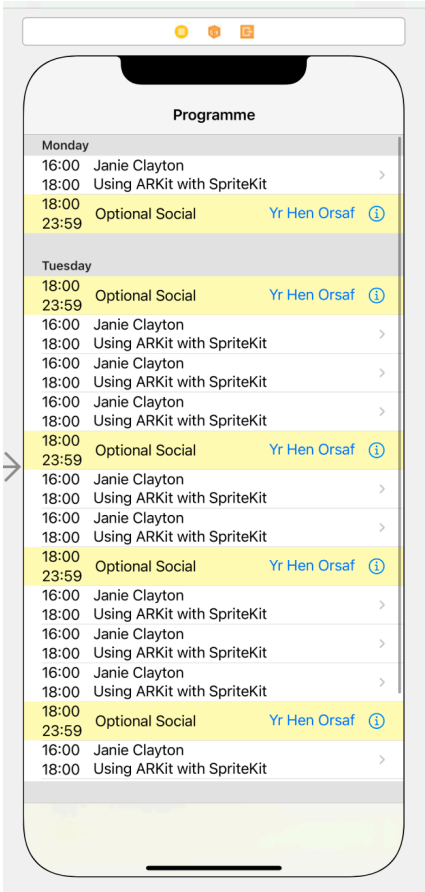
Step 16

- Add 4 labels to two types of custom table cell as shown (one cell for a talk slot, one for a break)
- Set background of break cell coloured yellow
- Add Accessory disclosure indicator to talk cell
- Add Accessory Detail to break cell



Step 17

- Having made the two types of cell, we can copy them, and paste many of them to make the correct structure for the Programme page
- We then need to change the data in the cells into representative data for our app



Finishing the app

- From here, there is nothing new to complete the prototype
- We add a talk screen with two labels, a textview and a button and we link the talk screen button to the speaker screen (or a copy of it with the correct data)
- We make the Favourites screen like the Programme screen but with less cells showing, and link it to the talk screen

We now have a complete set of app screens that we can try out on our users

We can gradually replace these fixed screens with real screens as we implement the app

30

Apple Human Interface guidelines

- Many of choices we have made might have been different on Android
- e.g. use of tabs and back buttons
- Apple have guidance for how to build apps on different platforms:
  - https://developer.apple.com/design/human-interface-guidelines/
- Worth reading to give apps native feel e.g. see navigation section for iOS

31

Session 6: Structures

- Creating structs and initialising them
- Properties and access
- Methods and using them
- Lab 6: Creating and using Structs

- This covers lesson 2.3 of Development in Swift Fundamentals

Unit 2—Lesson 3: Structures

Structures

```
struct Person {
  var name: String
}
```

Capitalize type names

Use lowercase for property names

Structures
Accessing property values

```
struct Person {
  var name: String
}

let person = Person(name: "Jasmine")
print(person.name)
```

```
Jasmine
```

Structures
Adding functionality

```
struct Person {
  var name: String

 func sayHello() {
    print("Hello there! My name is \(name)!")
  }

}

let person = Person(name: "Jasmine")
person.sayHello()
```

```
Hello there! My name is Jasmine!
```

Instances

```
struct Shirt {
  var size: String
  var color: String
}

let myShirt = Shirt(size: "XL", color: "blue")

let yourShirt = Shirt(size: "M", color: "red")
```

```
struct Car {
  var make: String
  var year: Int
  var color: String

  func startEngine() {...}

  func drive() {...}

  func park() {...}

  func steer(direction: Direction) {...}
}

let firstCar = Car(make: "Honda", year: 2010, color: "blue")
let secondCar = Car(make: "Ford", year: 2013, color: "black")

firstCar.startEngine()
firstCar.drive()
```

Initializers

```
let string = String.init() // ""
let integer = Int.init() // 0
let bool = Bool.init() // false
```

Initializers

```
var string = String() // ""
var integer = Int() // 0
var bool = Bool() // false
```

Initializers
Default values

```
struct Odometer {
  var count: Int = 0
}

let odometer = Odometer()
print(odometer.count)
```

```
0
```

Initializers
Memberwise initializers

```
let odometer = Odometer(count: 27000)
print(odometer.count)
```

```
27000
```

Initializers
Memberwise initializers

```
struct Person {
  var name: String
}
```

Initializers
Memberwise initializers

```
struct Person {
  var name: String

  func sayHello() {
    print("Hello there!")
  }
}

let person = Person(name: "Jasmine") // Memberwise initializer
```

```
struct Shirt {
  let size: String
  let color: String
}

let myShirt = Shirt(size: "XL", color: "blue") // Memberwise initializer


struct Car {
  let make: String
  let year: Int
  let color: String
}

let firstCar = Car(make: "Honda", year: 2010, color: "blue") // Memberwise initializer
```

Initializers
Custom initializers

```
struct Temperature {
  var celsius: Double
}

let temperature = Temperature(celsius: 30.0)
```

```
let fahrenheitValue = 98.6
let celsiusValue = (fahrenheitValue - 32) / 1.8

let newTemperature = Temperature(celsius: celsiusValue)
```

```
struct Temperature {
  var celsius: Double

  init(celsius: Double) {
    self.celsius = celsius
  }

  init(fahrenheit: Double) {
    celsius = (fahrenheit - 32) / 1.8
  }
}

let currentTemperature = Temperature(celsius: 18.5)
let boiling = Temperature(fahrenheit: 212.0)

print(currentTemperature.celsius)
print(boiling.celsius)

18.5
100.0
```

Unit 2—Lesson 3
Lab: Structures

Open and complete the exercises on page 1 of Lab – `Structures.playground`:

- Exercise - Structs, Instances, and Default Values

Instance methods

```
struct Size {
  var width: Double
  var height: Double

  func area() -> Double {
    return width * height
  }
}

var someSize = Size(width: 10.0, height: 5.5)

let area = someSize.area() // Area is assigned a value of 55.0
```

Mutating methods

```
struct Odometer {
  var count: Int = 0 // Assigns a default value to the 'count' property.
}
```

Need to

- Increment the mileage
- Reset the mileage

```
struct Odometer {
  var count: Int = 0 // Assigns a default value to the 'count' property.

  mutating func increment() {
    count += 1
  }

  mutating func increment(by amount: Int) {
    count += amount
  }

  mutating func reset() {
    count = 0
  }
}

var odometer = Odometer() // odometer.count defaults to 0
odometer.increment() // odometer.count is incremented to 1
odometer.increment(by: 15) // odometer.count is incremented to 16
odometer.reset() // odometer.count is reset to 0
```

Computed properties

```
struct Temperature {
  let celsius: Double
  let fahrenheit: Double
  let kelvin: Double
}

let temperature = Temperature(celsius: 0, fahrenheit: 32, kelvin: 273.15)
```

```
struct Temperature {
  var celsius: Double
  var fahrenheit: Double
  var kelvin: Double

  init(celsius: Double) {
    self.celsius = celsius
    fahrenheit = celsius * 1.8 + 32
    kelvin = celsius + 273.15
  }

  init(fahrenheit: Double) {
    self.fahrenheit = fahrenheit
    celsius = (fahrenheit - 32) / 1.8
    kelvin = celsius + 273.15
  }

  init(kelvin: Double) {
    self.kelvin = kelvin
    celsius = kelvin - 273.15
    fahrenheit = celsius * 1.8 + 32
  }
}

let currentTemperature = Temperature(celsius: 18.5)
let boiling = Temperature(fahrenheit: 212.0)
let freezing = Temperature(kelvin: 273.15)
```

```
struct Temperature {
  var celsius: Double

  var fahrenheit: Double {
    return celsius * 1.8 + 32
  }
}

let currentTemperature = Temperature(celsius: 0.0)
print(currentTemperature.fahrenheit)

32.0
```

Challenge

Add support for Kelvin

Modify the following to allow the temperature to be read as Kelvin

```
struct Temperature {
  let celsius: Double

  var fahrenheit: Double {
    return celsius * 1.8 + 32
  }

}
```

Hint: Temperature in Kelvin is Celsius + 273.15

```
struct Temperature {
  let celsius: Double

  var fahrenheit: Double {
    return celsius * 1.8 + 32
  }

  var kelvin: Double {
    return celsius + 273.15
  }

}

let currentTemperature = Temperature(celsius: 0.0)
print(currentTemperature.kelvin)

273.15
```

Property observers

```
struct StepCounter {
    var totalSteps: Int = 0 {
        willSet {
            print("About to set totalSteps to \(newValue)")
        }
        didSet {
            if totalSteps > oldValue  {
                print("Added \(totalSteps - oldValue) steps")
            }
        }
    }
}
```

Property observers

```
var stepCounter = StepCounter()
stepCounter.totalSteps = 40
stepCounter.totalSteps = 100
```

```
About to set totalSteps to 40
Added 40 steps
About to set totalSteps to 100
Added 60 steps
```

Type properties and methods

```
struct Temperature {
  static var boilingPoint = 100.0

  static func convertedFromFahrenheit(_ temperatureInFahrenheit: Double) -> Double {
    return(((temperatureInFahrenheit - 32) * 5) / 9)
  }

}

let boilingPoint = Temperature.boilingPoint

let currentTemperature = Temperature.convertedFromFahrenheit(99)

let biggestInt = Int.max
```

Copying

```
var someSize = Size(width: 250, height: 1000)
var anotherSize = someSize

someSize.width = 500

print(someSize.width)
print(anotherSize.width)
```

```
500
250
```

self

```
struct Car {
  var color: Color

  var description: String {
    return "This is a \(self.color) car."
  }
}
```

self
When not required

Not required when property or method names exist on the current object

```
struct Car {
  var color: Color

  var description: String {
    return "This is a \(color) car."
  }
}
```

self
When required

```
struct Temperature {
  var celsius: Double

  init(celsius: Double) {
    self.celsius = celsius
  }
}
```

Unit 2—Lesson 3

Lab: Structures

Open again and complete exercises on pages 3 and 5 of
`Lab - Structures.playground`

Session 7: Classes

- Inheritance in Swift
- Initializers
- Methods and properties
- Overriding methods and properties
- Difference between classes and structs
- When to use classes and structs
- Lab 7: Writing classes

- This covers lesson 2.4 of Development in Swift Fundamentals

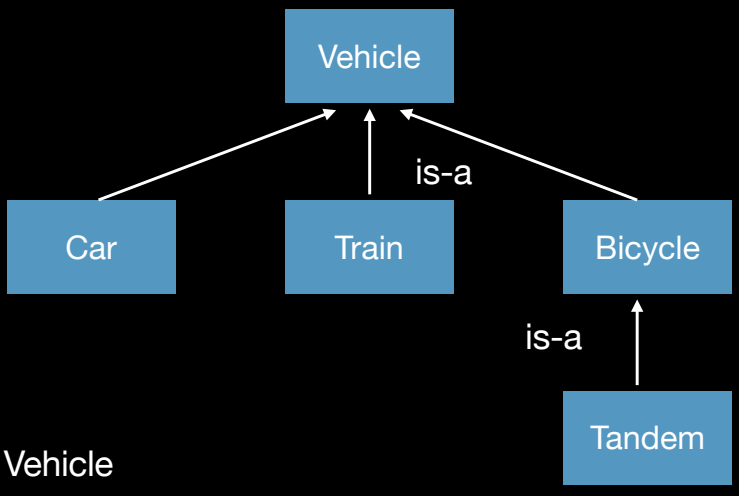
1

Unit 2—Lesson 4: Classes, Inheritance

```swift
class Person {
  let name: String

  init(name: String) {
    self.name = name
  }

  func sayHello() {
    print("Hello there!")
  }
}

let person = Person(name: "Jasmine")
print(person.name)
person.sayHello()

Jasmine
Hello there!
```

Inheritance

Base class



Car is Subclass of Vehicle

Vehicle is Superclass of Car

Inheritance
Defining a base class

```
class Vehicle {
   var currentSpeed = 0.0

   var description: String {
      return "traveling at \(currentSpeed) miles per hour"
   }

   func makeNoise() {
      // do nothing - an arbitrary vehicle doesn't necessarily make a noise
   }
}

let someVehicle = Vehicle()
print("Vehicle: \(someVehicle.description)")
```

```
Vehicle: traveling at 0.0 miles per hour
```

Inheritance
Create a subclass

```
class SomeSubclass: SomeSuperclass {
    // subclass definition goes here
}
```

```
class Bicycle: Vehicle {
    var hasBasket = false
}
```

Inheritance
Create a subclass

```
class Bicycle: Vehicle {
    var hasBasket = false
}

let bicycle = Bicycle()
bicycle.hasBasket = true

bicycle.currentSpeed = 15.0
print("Bicycle: \(bicycle.description)")
```

```
Bicycle: traveling at 15.0 miles per hour
```

Inheritance
Create a subclass

```
class Tandem: Bicycle {
    var currentNumberOfPassengers = 0
}
```

Inheritance
Create a subclass

```
class Tandem: Bicycle {
    var currentNumberOfPassengers = 0
}

let tandem = Tandem()
tandem.hasBasket = true
tandem.currentNumberOfPassengers = 2
tandem.currentSpeed = 22.0
print("Tandem: \(tandem.description)")
```

```
Tandem: traveling at 22.0 miles per hour
```

Inheritance
Override methods and properties

```
class Train: Vehicle {
    override func makeNoise() {
        print("Choo Choo!")
    }
}

let train = Train()
train.makeNoise()
```

```
Choo Choo!
```

Inheritance
Override methods and properties

```
class Car: Vehicle {
    var gear = 1
    override var description: String {
        return super.description + " in gear \(gear)"
    }
}
```

Inheritance
Override methods and properties

```
class Car: Vehicle {
    var gear = 1
    override var description: String {
        return super.description + " in gear \(gear)"
    }
}

let car = Car()
car.currentSpeed = 25.0
car.gear = 3
print("Car: \(car.description)")
```

```
Car: traveling at 25.0 miles per hour in gear 3
```

Inheritance
Override initializer

```
class Person {
  let name: String

  init(name: String) {
    self.name = name
  }
}

class Student: Person {
  var favoriteSubject: String
}
```

Class ‘Student’ has no initializers

Inheritance
Override initializer

```
class Person {
  let name: String

  init(name: String) {
    self.name = name
  }
}

class Student: Person {
  var favoriteSubject: String
  init(name: String, favoriteSubject: String) {
    self.favoriteSubject = favoriteSubject
    super.init(name: name)
  }
}
```

References

- When you create an instance of a class:
  - Swift returns the address of that instance
  - The returned address is assigned to the variable
- When you assign the address of an instance to multiple variables:
  - Each variable contains the same address
  - Update one instance, and all variables refer to the updated instance

```
class Person {
  let name: String
  var age: Int

  init(name: String, age: Int) {
    self.name = name
    self.age = age
  }
}

var jack = Person(name: "Jack", age: 24)
var myFriend = jack

jack.age += 1

print(jack.age)
print(myFriend.age)

25
25
```

```swift
struct Person {
  let name: String
  var age: Int
}

var jack = Person(name: "Jack", age: 24)
var myFriend = jack

jack.age += 1

print(jack.age)
print(myFriend.age)
```

```
25
24
```

Memberwise initializers

- Swift does not create memberwise initializers for classes
- Common practice is for developers to create their own for their defined classes

Class or structure?

- Start new types as structures
- Use a class:
  - When you're working with a framework that uses classes
  - When you want to refer to the same instance of a type in multiple places
  - When you want to model inheritance

Unit 2, Lesson 4

Lab: Classes.playground

Open and complete the exercises on pages 1 and 2 in `Lab – Classes.playground`

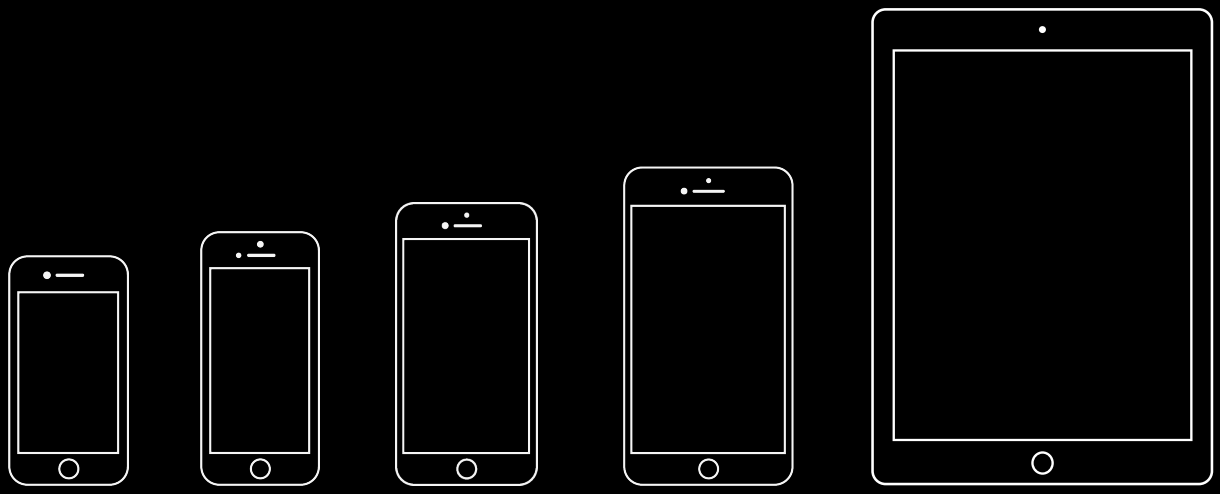
Session 8: Working with different screen sizes

- Autolayout
- Stack layout
- Lab 8: Fitting many items on different size screens
- Further exercises on Autolayout
- This covers lesson 2.10 of Development in Swift Fundamentals

1

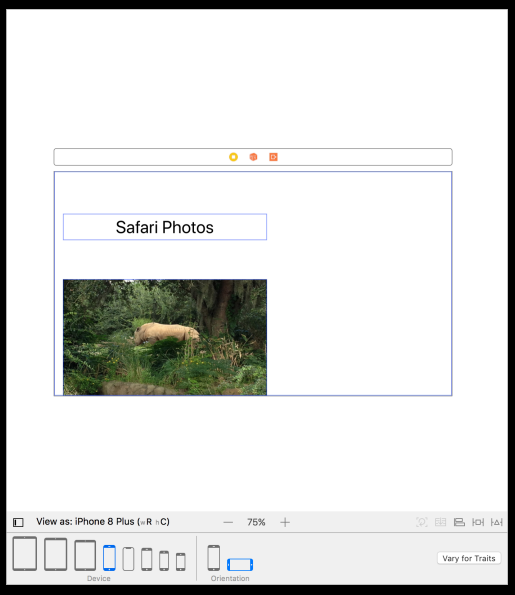
Lesson 2—10:
Auto Layout and Stack Views

Layout for multiple sizes

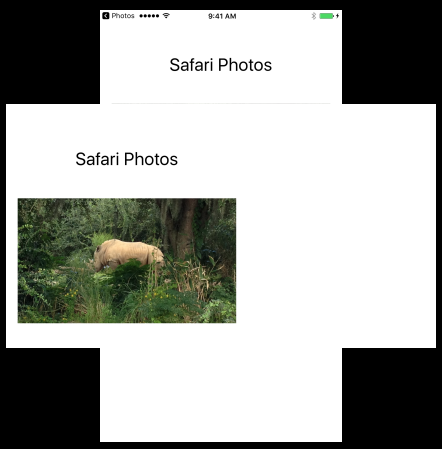


Interface Builder

View as:

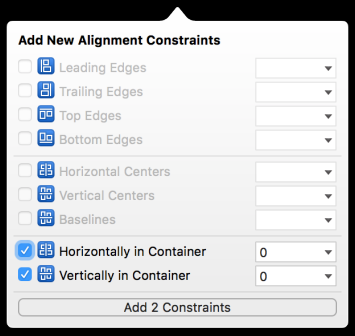


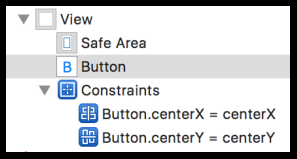
Why Auto Layout?

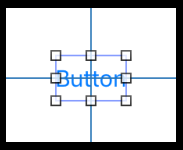


Create alignment constraints

Text

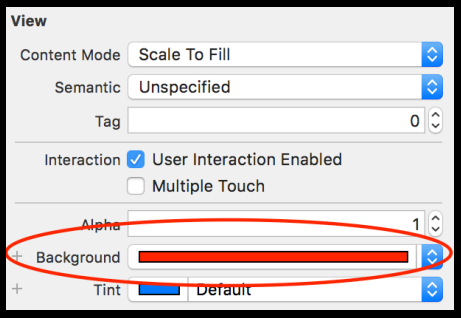


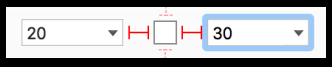




Create size constraints

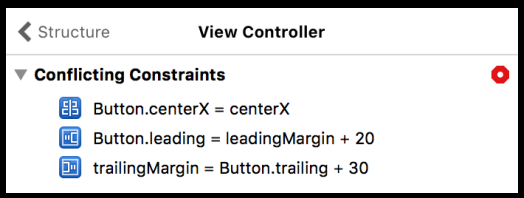
Text

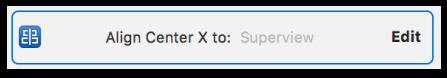




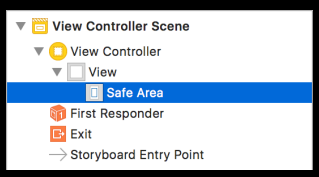
Resolve constraint issues

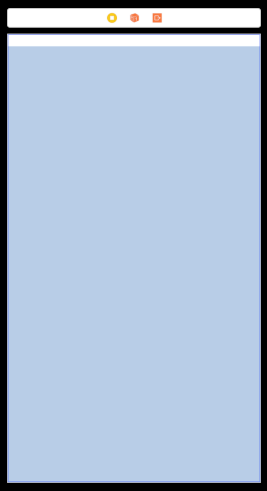
Text





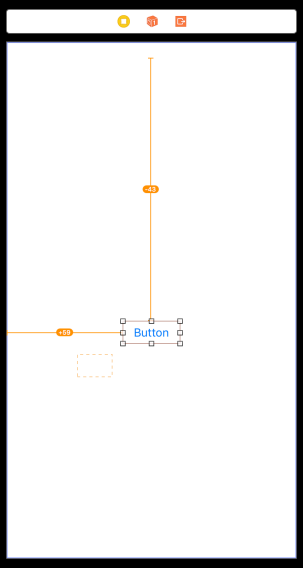
Safe area layout guide

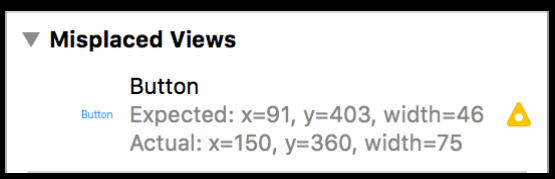




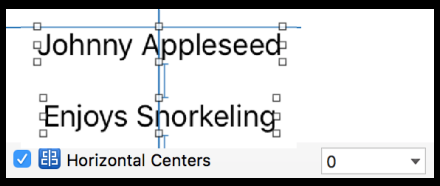
Resolve constraint warnings

Text





Constraints between siblings

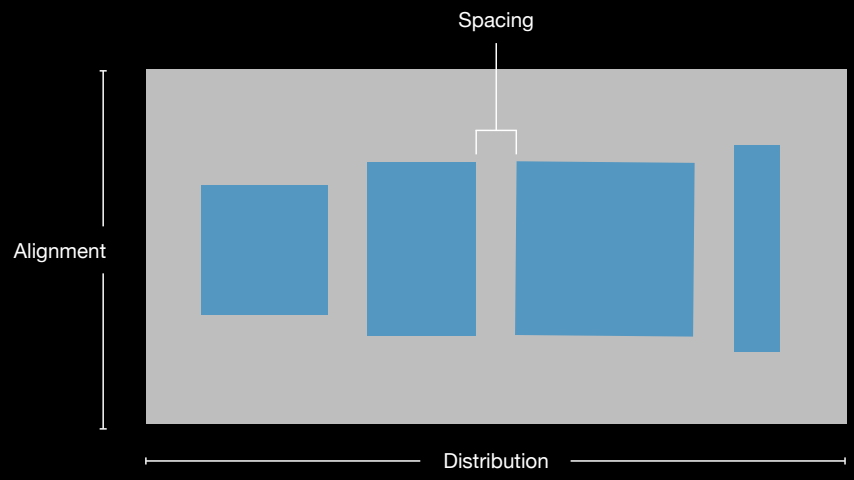


Stack views

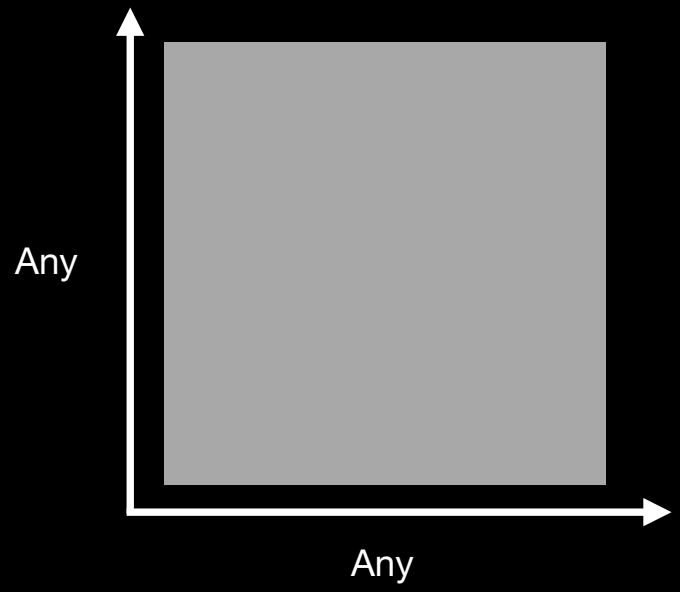
Text

Stack views

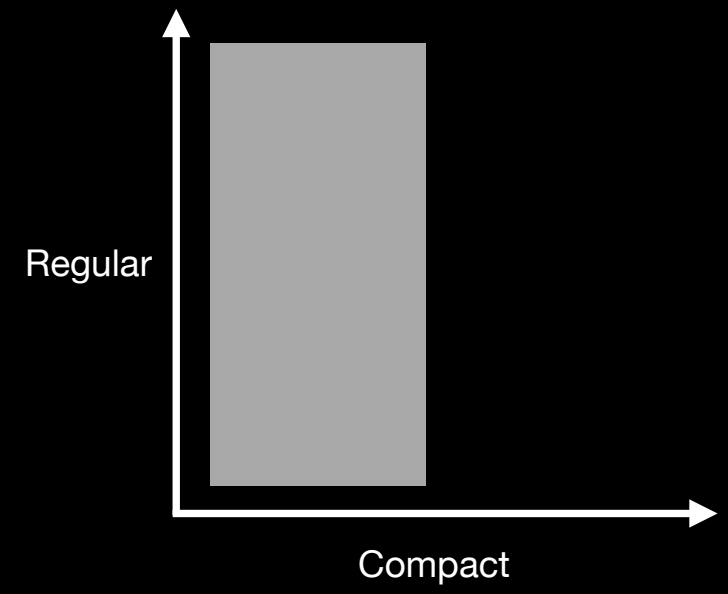
Stack view attributes



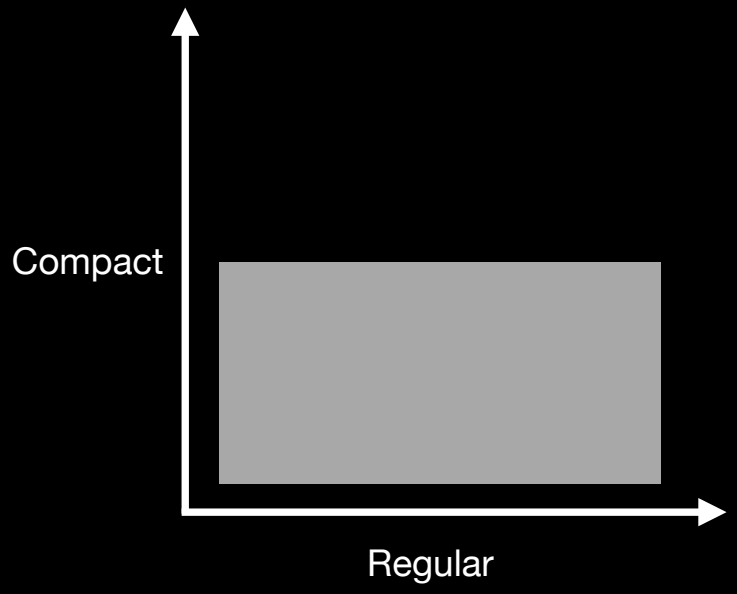
Size classes



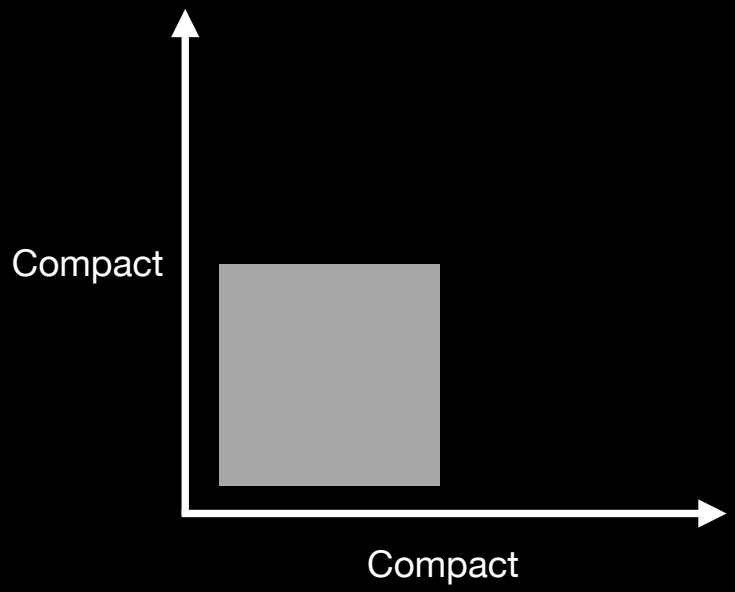
Size classes



Size classes

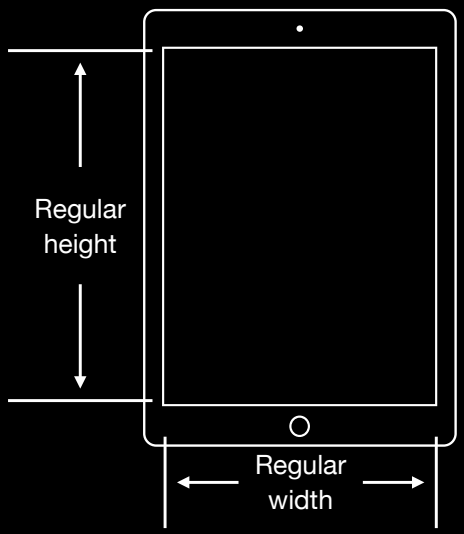


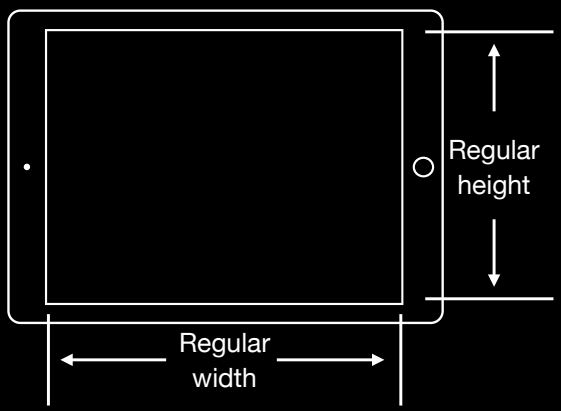
Size classes



Size classes

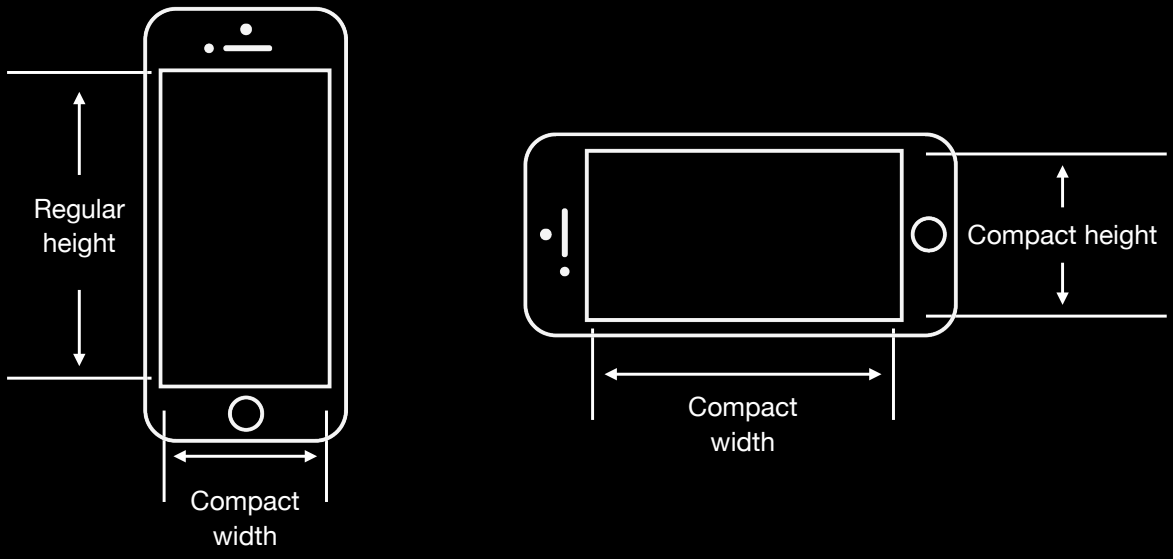
iPad





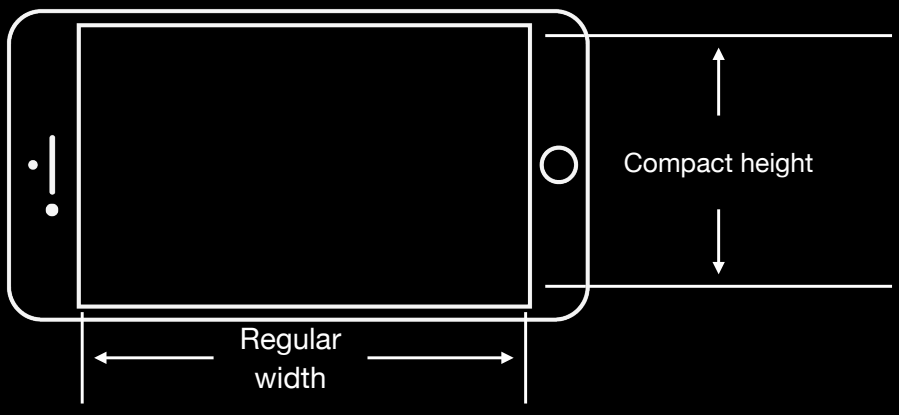
Size classes

iPhone



Size classes

iPhone 6 Plus, iPhone 7 Plus, and iPhone 8 Plus

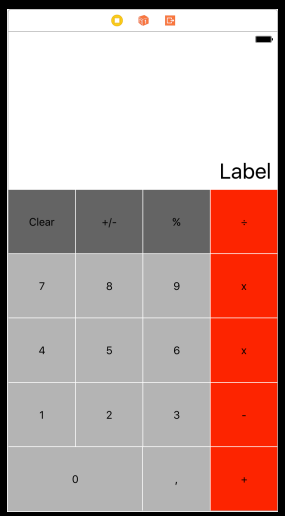


Unit 2—Lesson 10
Auto Layout and Stack Views

Learn the fundamentals of Auto Layout for building precisely designed user interfaces.

Unit 2—Lesson 10
Lab: Calculator

Use view objects, constraints, and stack views to create a simple calculator that maintains its layout on all device sizes.



Session 9: Dictionaries/Optionals / Guard / Scope

- Dictionaries/why we need optionals
- How to use them correctly
- Making code tidy with Guard
- Understanding scope of variables
- Lab 9: Optionals and Guard

- This covers lessons 2.5, 3.1, 3.3, 3.4 of Development in Swift Fundamentals

1

Unit 2—Lesson 5: Collections

(We have done Arrays, so will just do Dictionaries, page 175)

Dictionaries

```
[key1 : value1, key2: value2, key3: value3]
```

```
var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]
```

```
var myDictionary = [String: Int]()
var myDictionary = Dictionary<String, Int>()
var myDictionary: [String: Int] = [:]
```

Add/remove/modify a dictionary

Adding or modifying

```
var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]

scores["Oli"] = 399

let oldValue = scores.updateValue(100, forKey: "Richard")
```

Add/remove/modify a dictionary
Adding or modifying

```
var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]

scores["Oli"] = 399

if let oldValue = scores.updateValue(100, forKey: "Richard") {
  print("Richard's old value was \(oldValue)")
}
```

```
Richard's old value was 500
```

Add/remove/modify a dictionary
Removing

```
var scores = ["Richard": 100, "Luke": 400, "Cheryl": 800]
scores["Richard"] = nil
print(scores)

if let oldValue = scores.removeValue(forKey: "Luke") {
  print("Luke's score was \(oldValue) before he stopped playing")
}
print(scores)
```

```
["Cheryl": 800, "Luke": 400]
Luke's score was 400 before he stopped playing
["Cheryl": 800]
```

Accessing a dictionary

```
var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]

let players = Array(scores.keys) //["Richard", "Luke", "Cheryl"]
let points = Array(scores.values) //[500, 400, 800]

if let myScore = scores["Luke"] {
  print(myScore)
}
```

```
400
```

```
if let henrysScore = scores["Henry"] {
  print(henrysScore)
}
```

Unit 2—Lesson 5
Lab: Collections

Open and complete exercise 3 (Dictionaries) in

`Lab - Collections.playground`

Unit 3—Lesson 1: Optionals

Last week we saw this stuff with Dictionaries
This is an example of Optionals

```
var scores = ["Richard": 500, "Luke": 400, "Cheryl": 800]

scores["Oli"] = 399

if let oldValue = scores.updateValue(100, forKey: "Richard") {
  print("Richard's old value was \(oldValue)")
}
```

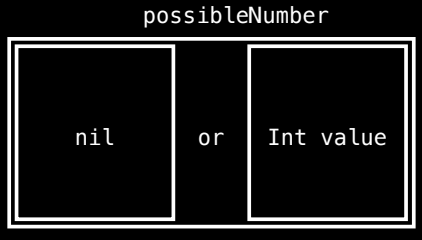
```
Richard's old value was 500
```

Functions and optionals
Return values

```
let numberString = "123"
let possibleNumber = Int(numberString)
```

```
let numberString = "Cynthia"
let possibleNumber = Int(numberString)
```

Type of Int(String) is an Optional Integer



Working with optional values

Force-unwrap - dangerous if value is nil

```
let possibleNumber = Int(numberString)
if possibleNumber != nil {
  let actualNumber = possibleNumber!
  print(actualNumber)
}

let unwrappedNumber = possibleNumber!
```

error: Execution was interrupted

Specifying the type of an optional

```
var serverResponseCode = 404
```

```
var serverResponseCode = nil
```

'nil' requires a contextual type

```
var serverResponseCode: Int? = 404
```

```
var serverResponseCode: Int? = nil
```

Working with optional values
Optional binding

```
if let constantName = someOptional {
  //constantName has been safely unwrapped for use within the braces.
}
```

```
if let possibleNumber = Int(numberString) {
  print("The value of the string was \(possibleNumber)")
}
else {
  print(“That was not a legal integer value”)
}
```

Functions and optionals
Defining

```
func printFullName(firstName: String, middleName: String?, lastName: String)
```

```
func textFromURL(url: URL) -> String?
```

Optional chaining

```
class Person {
  var age: Int
  var residence: Residence?
}

class Residence {
  var address: Address?
}

class Address {
  var buildingNumber: String?
  var streetName: String?
  var apartmentNumber: String?
}
```

Optional chaining

```
if let theResidence = person.residence {
  if let theAddress = theResidence.address {
    if let theApartmentNumber = theAddress.apartmentNumber {
      print("He/she lives in apartment number \(theApartmentNumber).")
    }
  }
}
```

Optional chaining

```
if let theApartmentNumber = person.residence?.address?.apartmentNumber {
  print("He/she lives in apartment number \(theApartmentNumber).")
}
```

Implicitly Unwrapped Optionals

```
class ViewController: UIViewController {
  @IBOutlet weak var label: UILabel!
}
```

Unwraps automatically

Should only be used when need to initialize an object without supplying the value and you'll be giving the object a value soon afterwards

Unit 3—Lesson 3: Guard

```
func singHappyBirthday() {
  if birthdayIsToday {
    if invitedGuests > 0 {
      if cakeCandlesLit {
        print("Happy Birthday to you!")
      } else {
        print("The cake candle's haven't been lit.")
      }
    } else {
      print("It's just a family party.")
    }
  } else {
    print("No one has a birthday today.")
  }
}
```

```
func singHappyBirthday() {
  guard birthdayIsToday else {
    print("No one has a birthday today.")
    return
  }

  guard invitedGuests > 0 else {
    print("It's just a family party.")
    return
  }

  guard cakeCandlesLit else {
    print("The cake's candles haven't been lit.")
    return
  }

  print("Happy Birthday to you!")
}
```

guard

```
guard condition else {
  //false: execute some code
}

//true: execute some code
```

guard

```
func divide(_ number: Double, by divisor: Double) {
  if divisor != 0.0 {
    let result = number / divisor
    print(result)
  }
}
```

```
func divide(_ number: Double, by divisor: Double) {
  guard divisor != 0.0 else { return }

  let result = number / divisor
  print(result)
}
```

guard with optionals

```
if let eggs = goose.eggs {
  print("The goose laid \(eggs.count) eggs.")
}
//`eggs` is not accessible here
```

```
guard let eggs = goose.eggs else { return }
//`eggs` is accessible hereafter
print("The goose laid \(eggs.count) eggs.")
```

guard with optionals

```
func processBook(title: String?, price: Double?, pages: Int?) {
  if let theTitle = title, let thePrice = price, let thePages = pages {
    print("\(theTitle) costs $\(thePrice) and has \(thePages) pages.")
  }
}
```

```
func processBook(title: String?, price: Double?, pages: Int?) {
  guard let theTitle = title, let thePrice = price, let thePages = pages else { return }
  print("\(theTitle) costs $\(thePrice) and has \(thePages) pages.")
}
```

Unit 3 — Lesson 4:
Constant and Variable Scope

Scope

Global scope—Defined outside of a function

Local scope—Defined within braces (`{` `}`)

```
var globalVariable = true

if globalVariable {
 let localVariable = 7
}
```

Scope

```
var age = 55

func printMyAge() {
  print("My age: \(age)")
}

print(age)
printMyAge()
```

```
55
My age: 55
```

Scope

```
func printBottleCount() {
    let bottleCount = 99
    print(bottleCount)
}

printBottleCount()
print(bottleCount)
```

Use of unresolved identifier 'bottleCount'

Scope

```
func printTenNames() {
  var name = "Richard"
  for index in 1...10 {
    print("\(index): \(name)")
  }
  print(index)
  print(name)
}

printTenNames()
```

Use of unresolved identifier 'index'

Variable shadowing

```
let points = 100

for index in 1...3 {
  let points = 200
  print("Loop \(index): \(points+index)")
}
print(points)
```

```
Loop 1: 201
Loop 2: 202
Loop 3: 203
100
```

Variable shadowing

```
var name: String? = "Robert"

if let name = name {
 print("My name is \(name)")
}
```

Variable shadowing

```
func exclaim(name: String?) {
  if let name = name {
    print("Exclaim function was passed: \(name)")
  }
}
```

```
func exclaim(name: String?) {
  guard let name = name else { return }
  print("Exclaim function was passed: \(name)")
}
```

Shadowing and initializers

```
struct Person {
  var name: String
  var age: Int
}

let todd = Person(name: "Todd", age: 50)
print(todd.name)
print(todd.age)
```

```
Todd
50
```

Shadowing and initializers

```
struct Person {
  var name: String
  var age: Int

  init(name: String, age: Int) {
    self.name = name
    self.age = age
  }
}
```

Unit 3, Lesson 1
Lab: Optionals.playground

Open and complete the exercises in `Lab - Optionals.playground`

Unit 3—Lesson 3
Lab: Guard

Open and complete the exercises in `Lab - Guard.playground`

Unit 3—Lesson 4
Lab: Constant and Variable Scope

Open and complete the exercises in `Lab - Scope.playground`

Session 10: Advanced navigation

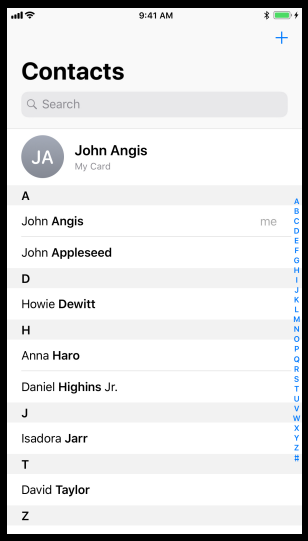
- Segues
- Navigation
- Tab bars
- Lifetime of an app
- *Lab 10: Making complex apps*

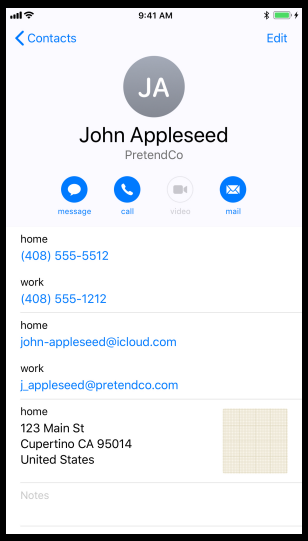
- This covers lessons 3.6, 3.7, 3.8 of Development in Swift Fundamentals

1

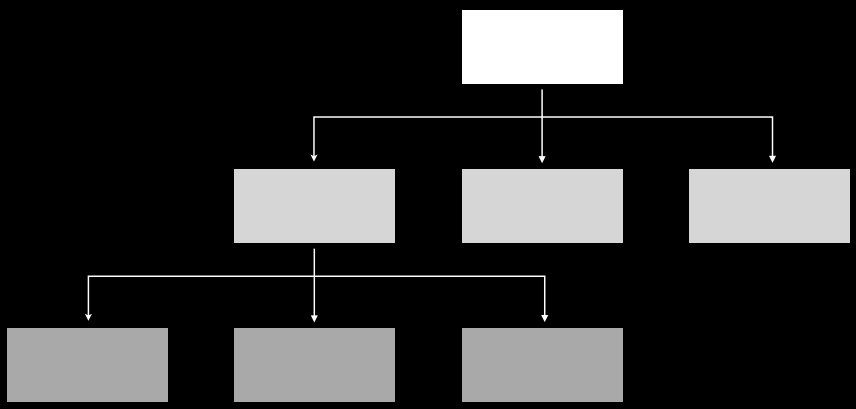
Unit 3—Lesson 6: Segues and Navigation Controllers

Segues and navigation controllers





Navigation hierarchy
Hierarchical



Segues (UIStoryboardSegue)

A `UIStoryboardSegue` object performs the visual transition between two view controllers

It is also used to prepare for the transition from one view controller to another

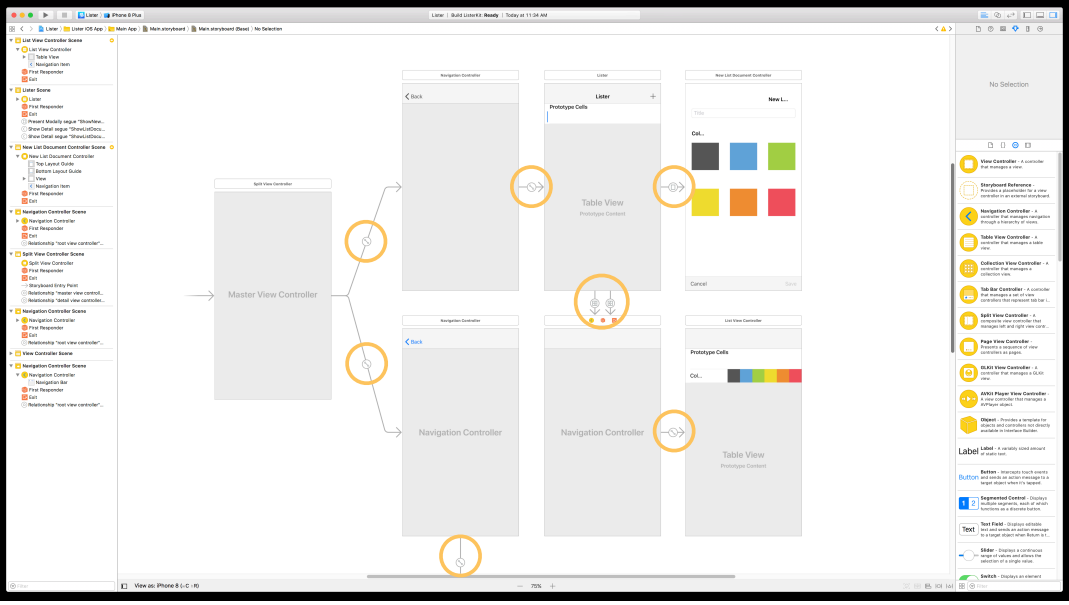
Segue objects contain information about the view controllers that are involved in a transition

When a segue is triggered, before the visual transition occurs, the storyboard runtime can call certain methods in the current view controller

Useful if you need to pass information forward

Segues (UIStoryboardSegue)

Segues between scenes



Segues (UIStoryboardSegue)
Unwind

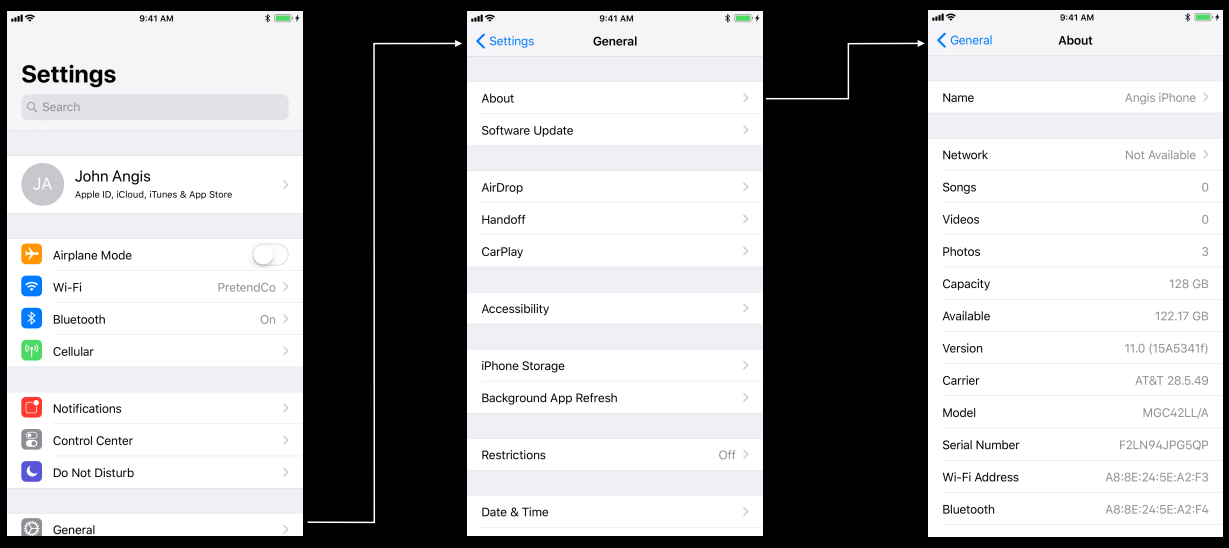
```
@IBAction func myUnwindFunction(unwindSegue: UIStoryboardSegue) {
}
```

Implement the returned method in the view controller you wish to return to

Doesn’t need to do anything apart from being implemented

Connect this to the view controller returning from

Navigation controller (UINavigationController)



Navigation controller

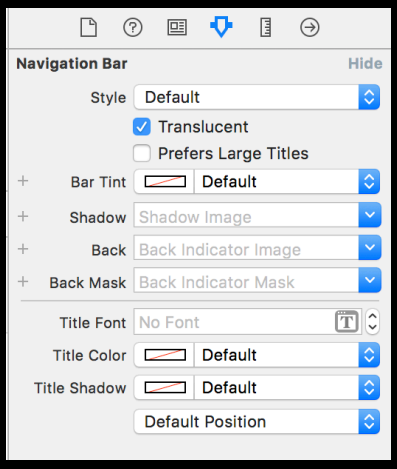
The top view controller's title

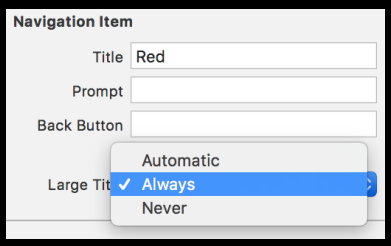
Back button

Navigation bar

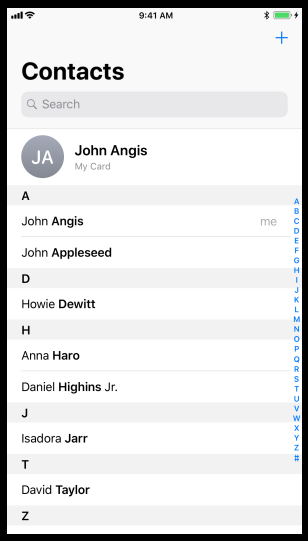
The top view controller's view

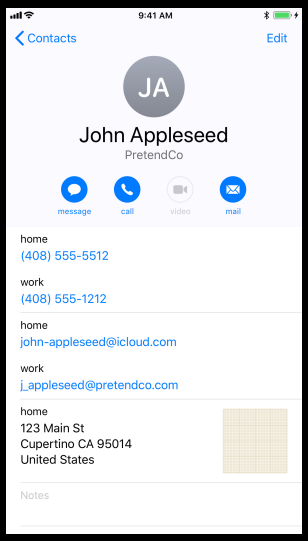
Navigation controller
Large titles





Pass information





Pass information

```
func prepare(for segue: UIStoryboardSegue, sender: Any?)
```

Segue properties

- `identifier`
- `destination`

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
   segue.destination.navigationItem.title = textField.text
}
```

Create programmatic segues

```
performSegue(withIdentifier:, sender:)
```

```
performSegue(withIdentifier: "ShowDetail", sender: nil)
```

Unit 3—Lesson 6
Segues and Navigation Controllers

Learn how to use segues to transition from one view controller to another

How to define relationships between view controllers

How navigation controllers can help you manage scenes that display related or hierarchical content

Unit 3—Lesson 6
Lab: Login

Create a login screen that will pass a user name between view controllers

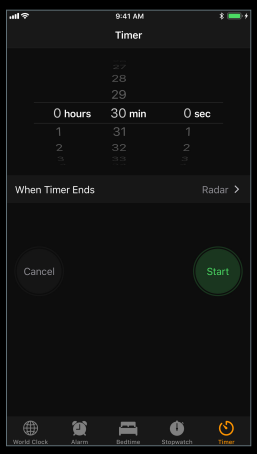
Use view controllers, a navigation controller, and segues to create both the login screen and a simple landing screen that will display in its title either the user name or text related to a forgotten user name or password

Unit 3—Lesson 7: Tab Bar Controllers

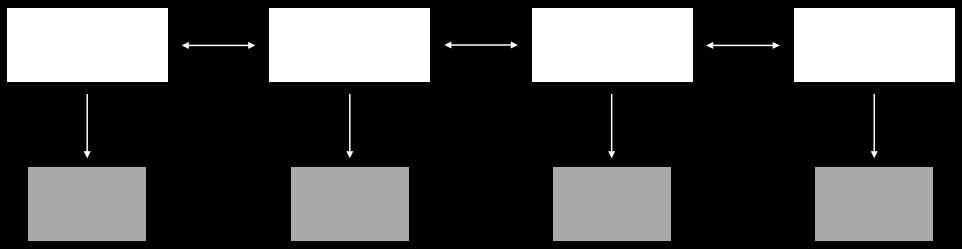
UITabBarController

A specialized view controller that manages a radio-style selection interface

A tab bar is displayed at the bottom of the view



Navigation hierarchy
Flat



UITabBarController
Configuration

Add a tab bar controller

Using a storyboard

Drag in a UITabBarController from the object library

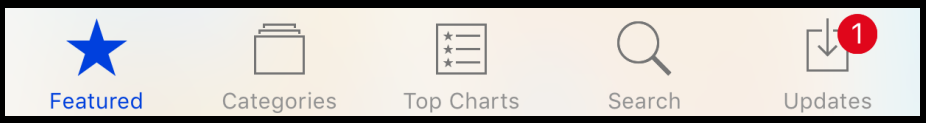
Add tabs

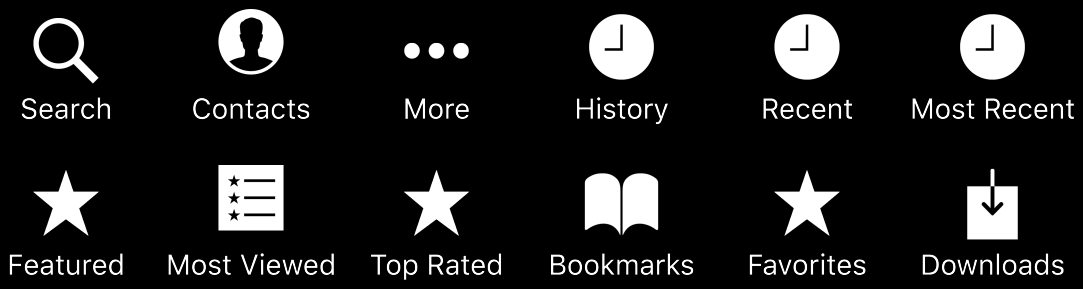
Drag a new view controller object onto the canvas

To create a segue, control-drag from the `UITabBarController` to the view controller

Select "view controllers" under Relationship Segue

UITabBarItem
Glyphish





Programmatic customization

```
tabBarItem.badgeValue = "!"
```

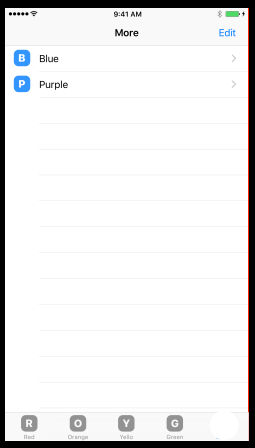

```
tabBarItem.badgeValue = nil
```

Even more tab items
More ! = Better

More view controller:

- Appears when needed
- Can't be customized

If possible, plan app to avoid More



Unit 3—Lesson 7
Tab Bar Controllers

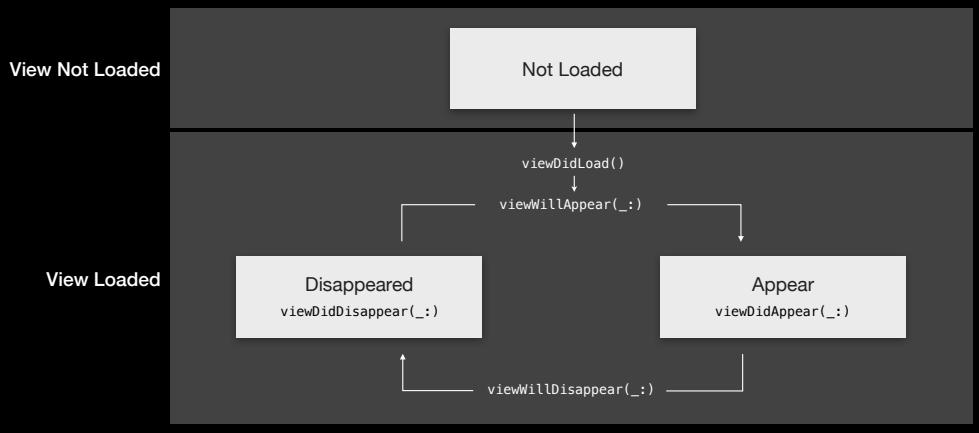
Learn how to use tab bar controllers to organize different kinds of information or different modes of operation.

Unit 3—Lesson 7
Lab: About Me

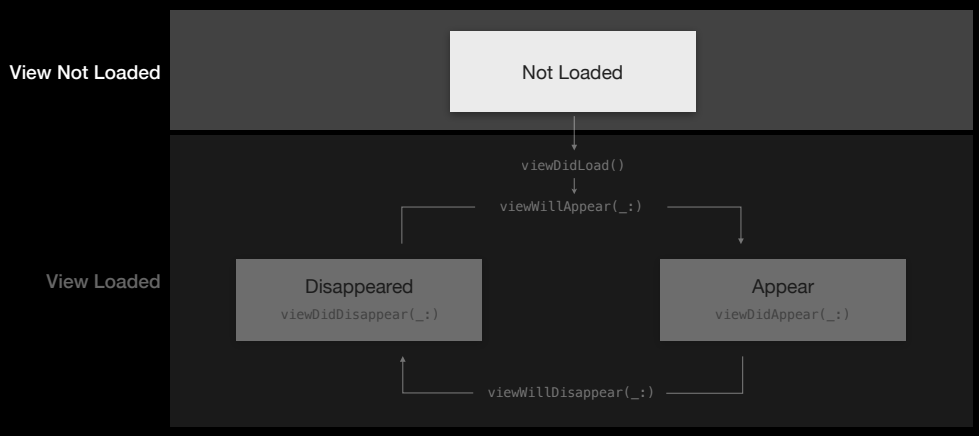
Create an app that displays distinct types of information about yourself in separate tabs.

Unit 3—Lesson 8:
View Controller Life Cycle

View controller life cycle



View controller life cycle
viewDidLoad( )

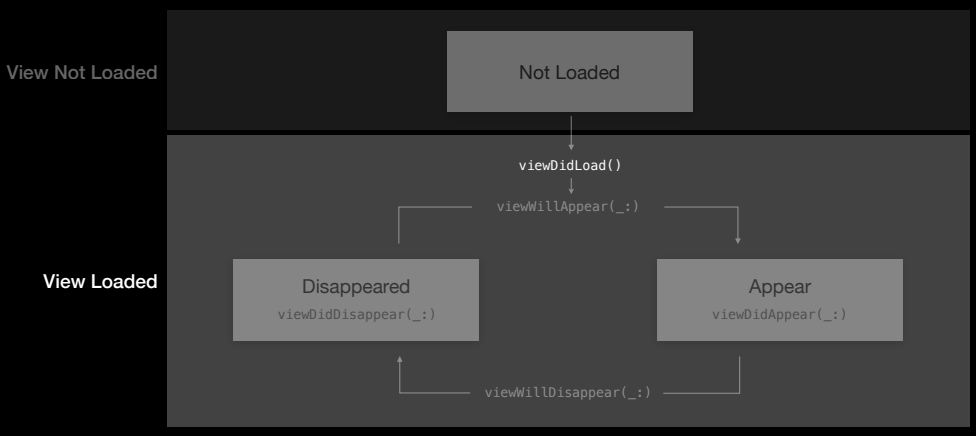


View event management

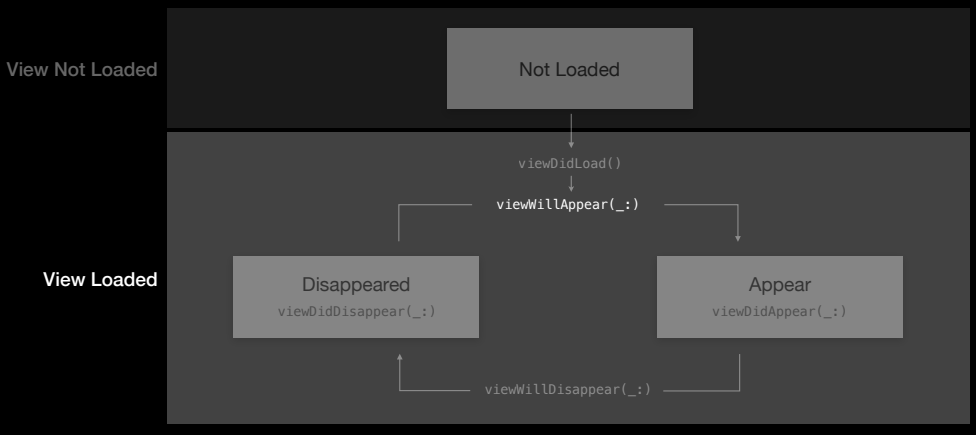
```
viewWillAppear(_:)
viewDidAppear(_:)
viewWillDisappear(_:)
viewDidDisappear(_:)
```

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    // Add your code here
}
```

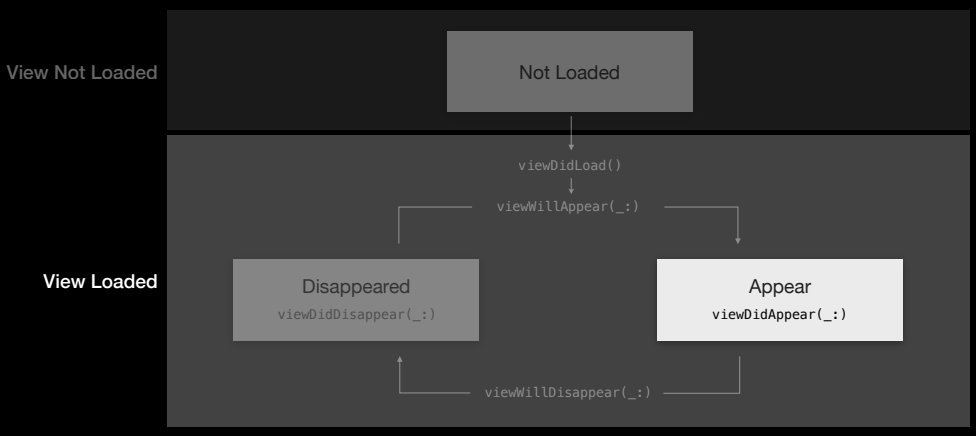
View event management
viewWillAppear(_:)



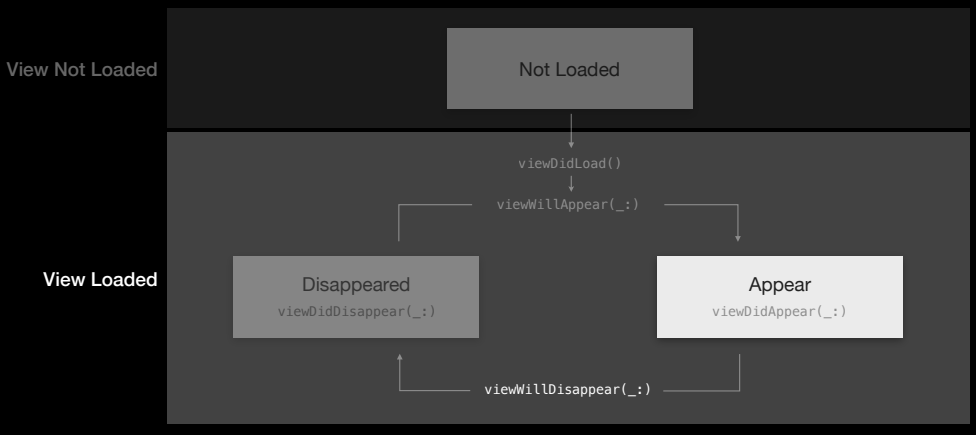
View event management
viewDidAppear(_:)



View event management
viewWillDisappear(_:)



View event management
viewDidDisappear(_:)



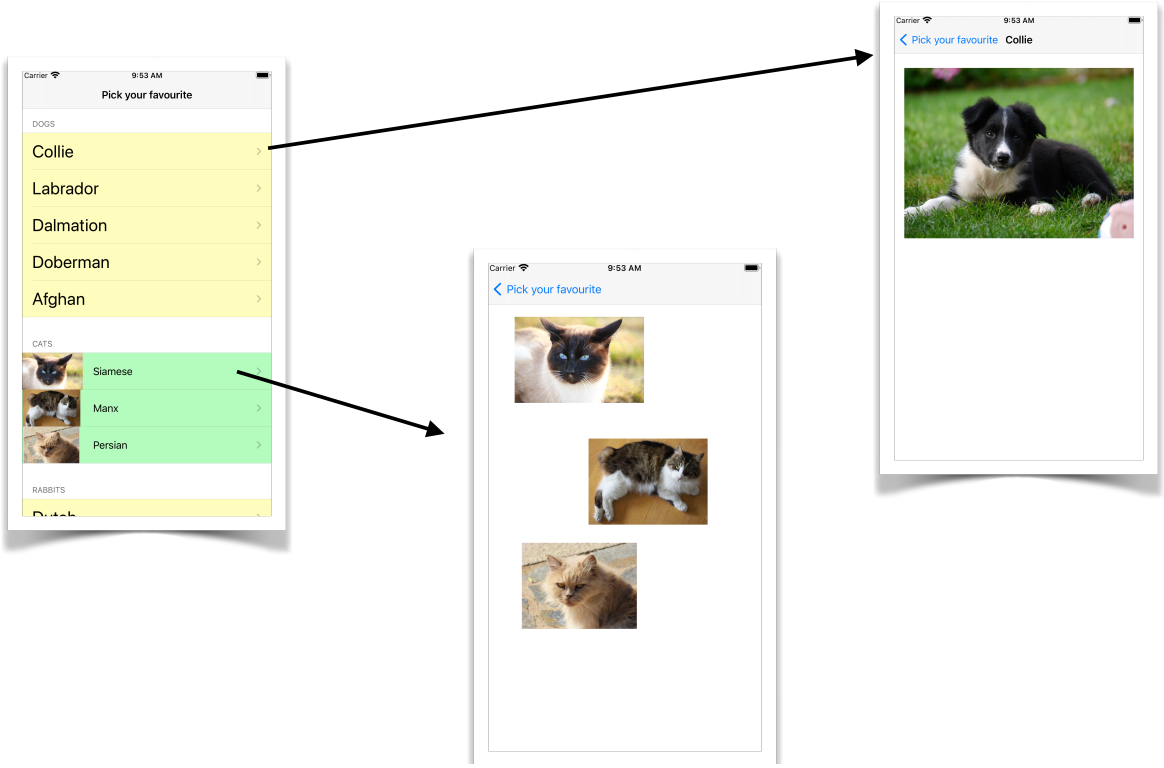
Unit 3—Lesson 8
View Controller Life Cycle

This lesson will explain more about the view controller life cycle so you can understand the infinite potential of this important class

Unit 3—Lesson 8
Lab: Order of Events

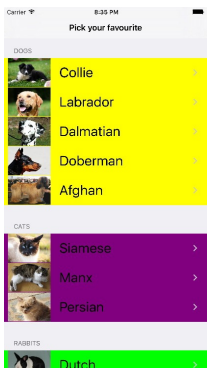
Further your understanding of the view's life cycle by creating an app that adds to a label's text based on the events in the view controller life cycle

Session 12: Making Dynamic Tables



Delegates - allow you to supply info

UITableViewController uses *delegates* to specify what to show in a table:



- How many SECTIONS are in the table?
- What are the headers/footers for each section?
- How tall is each header/footer?
- How many cells are in each section?
- What does each cell in each section look like?

There are default answers for each of these. If you don't want the default, you must provide a delegate.

There are two sets of delegates associated with a table:

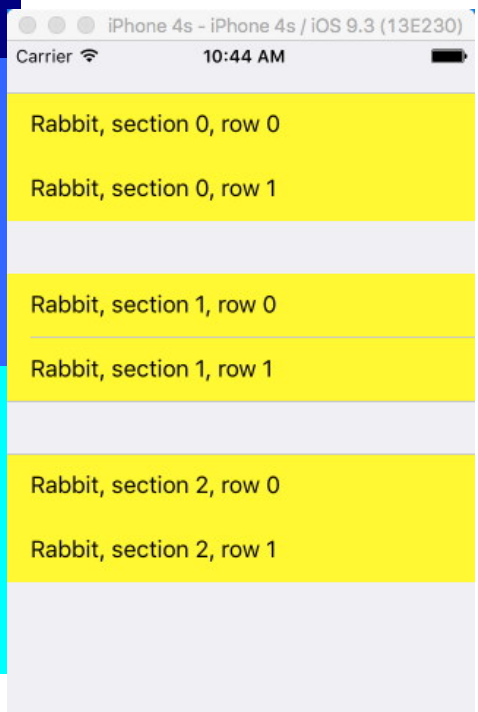
- UITableViewDataSource has delegates that provide details of what data to use in the table
- UITableViewDelegate has other delegates that you customise to say what the table looks like

Minimum that you need to define is:

- How many sections there are
- How many rows are in each section
- What text is in each row in each section

3

First Animal App - Minimum answers for a table



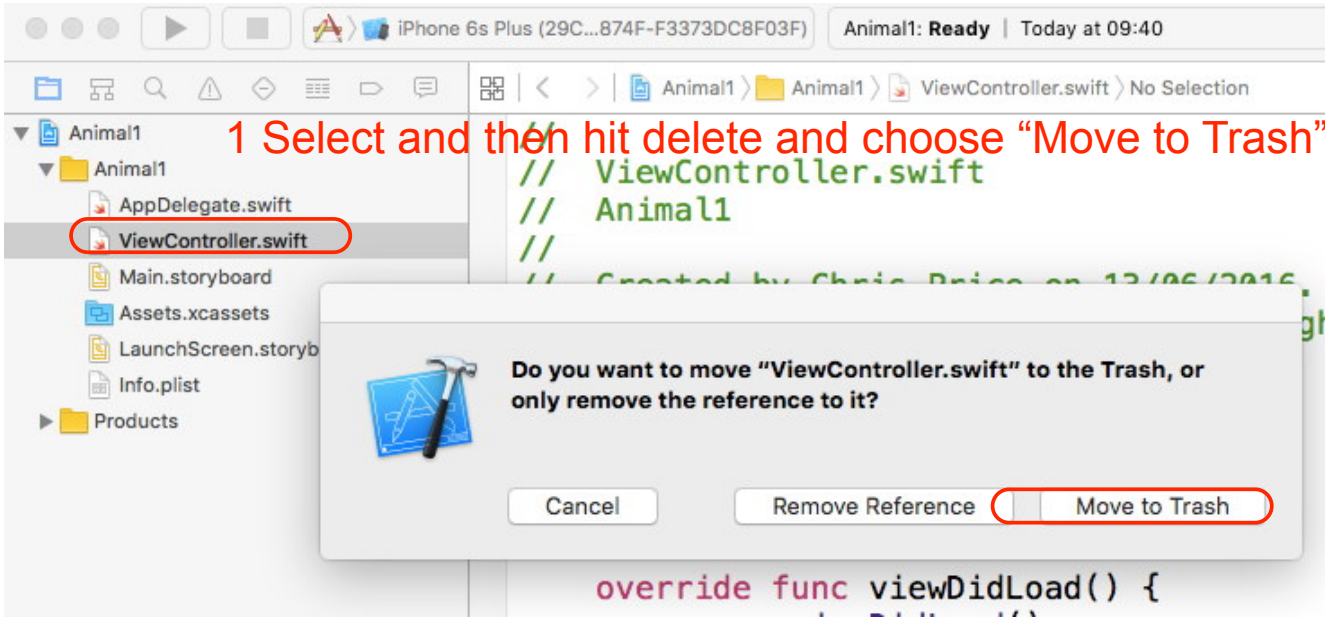
Step 1

•Make a new Project called Animal1
•Make it a Single View project

4

Step 2

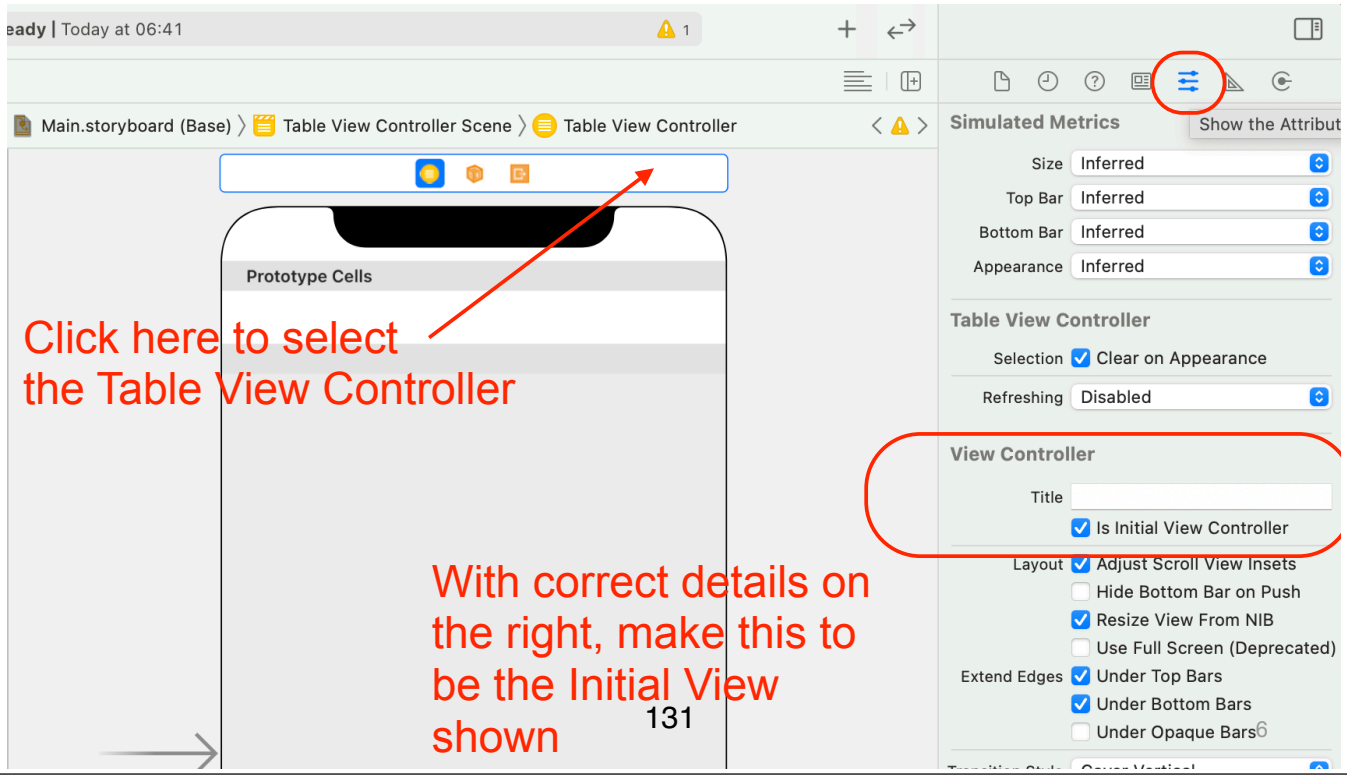
•Delete ViewController.swift as we won't be using it.
•Go to Main.storyboard, select the window and delete it (as we did when making static tables).



2 Select storyboard and get rid of the ViewController view

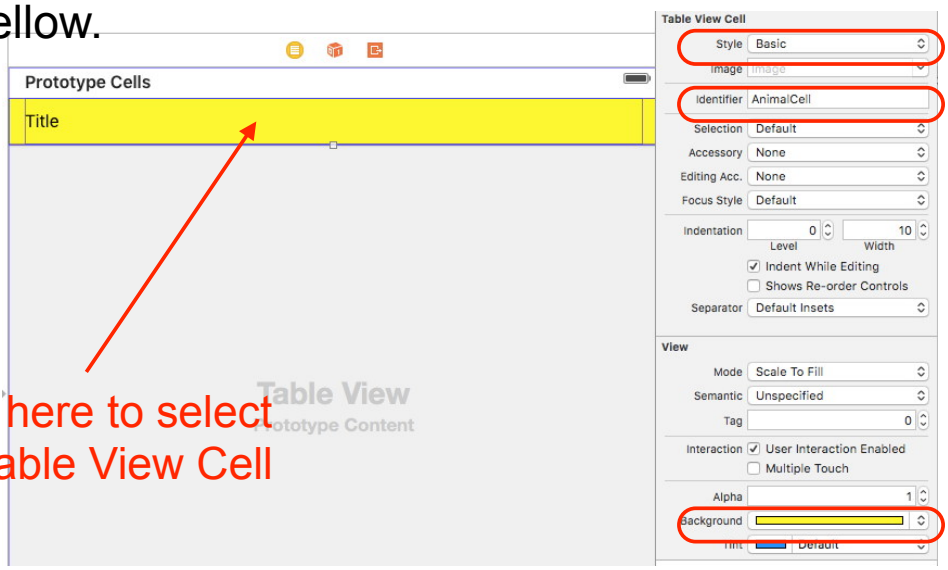
Step 3

- Add a new TableViewController screen to the story board.
- Select it and choose to make it initial view controller.



Step 4

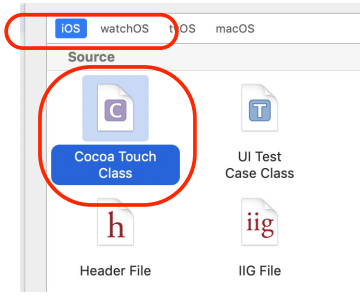
- Select the Table View Cell. Make its style Basic, make its identifier AnimalCell, and make its background colour yellow.



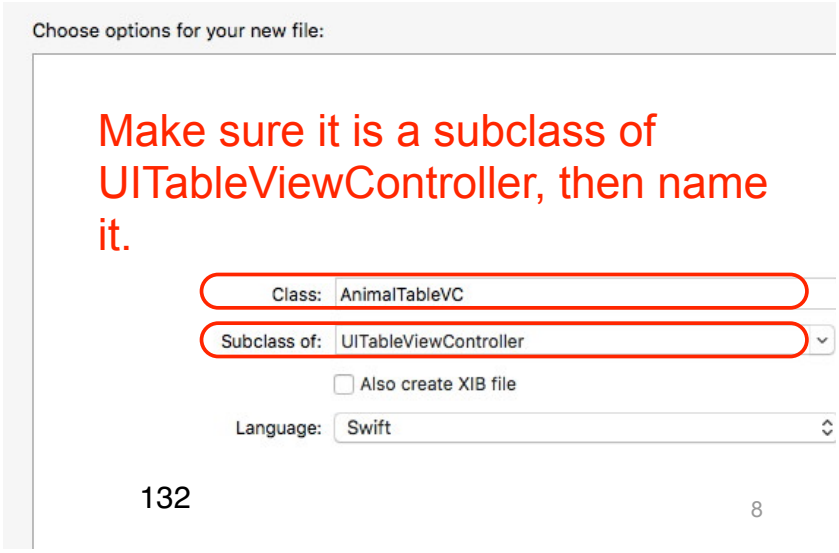
- You can run your app now, but it will look pretty dull as you don't have any code associated with the View.
- You now need to create a UITableViewController file and write some code to define what goes in the table.

Step 5

•Select File/New/File to be asked for a template for the file.
•Choose iOS/Source/Cocoa Touch Class, and click Next to be asked "Choose options for your file".
•We are going to call our class AnimalTableVC and base ir on UITableViewController, then hit Next a couple of times to create it.



Choose the right kind of file



Step 6

The template for a UITableViewController contains outline definitions for several functions:

- viewDidLoad - called when the window starts the first time. Good place to set things up. *Leave for now.*
- numberOfSectionsInTableView - This function returns how many sections there are in the table. *Make it return 3 instead of zero.*
- tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int
- This function tells you for which section it wants to know how many rows it has. *Make it 2 for each section.*

Step 7

UITableViewController also has some commented out functions. Uncomment the first of those

- tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: IndexPath) -> UITableViewCell
- This delegate is told the section and row of the cell that is wanted (indexPath.section and indexPath.row) and has to return a UITableCell that defines what the cell should look like.

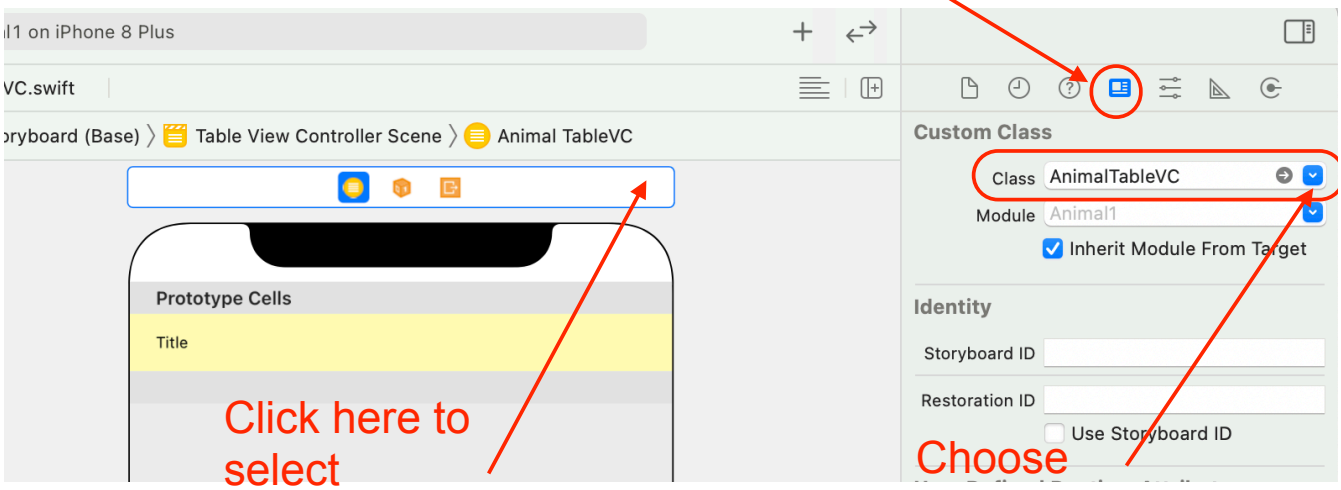
Make cellForRowAtIndexPath look like this:

```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "AnimalCell",
                                             for: indexPath)
    // Configure the cell...
    cell.textLabel?.text = "Rabbit, section \(indexPath.section), row \(indexPath.row)"
    return cell
}
```

Step 8

•We still need to tell the View on the storyboard that this new class AnimalTableVC tells it how to prepare its content.

Need the Identity Inspector instead of the Attribute Inspector to see the right dialog.



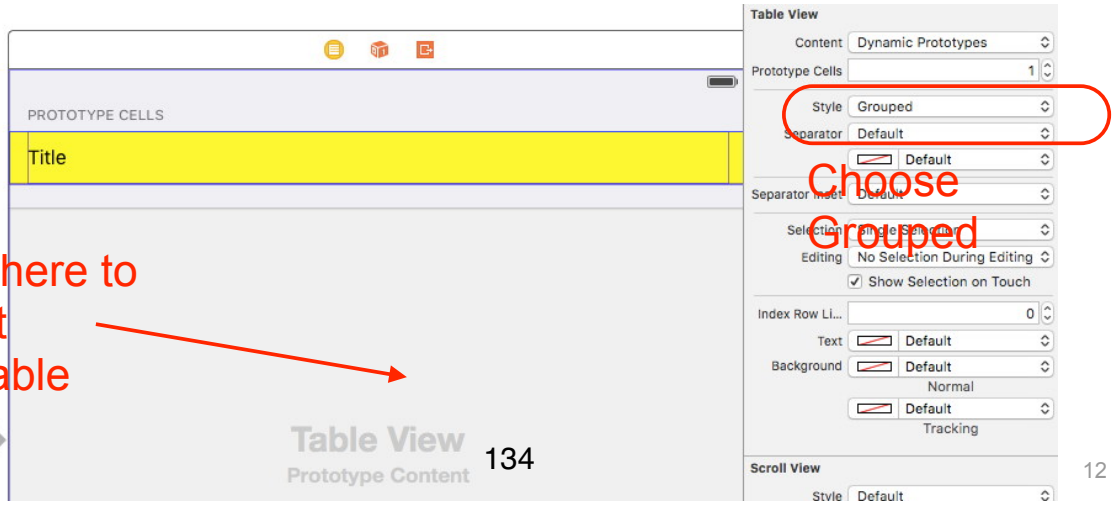
Click here to select the TableView Controller screen

Choose AnimalTableVC from drop-down menu

Finished But!

•If you run it now, it should generate content for 2 cells in each of 3 sections (note that sections and cells are numbered from zero)

•The section split can be made more obvious by selecting the Table in the story board, and then changing the style to Grouped instead of Plain.

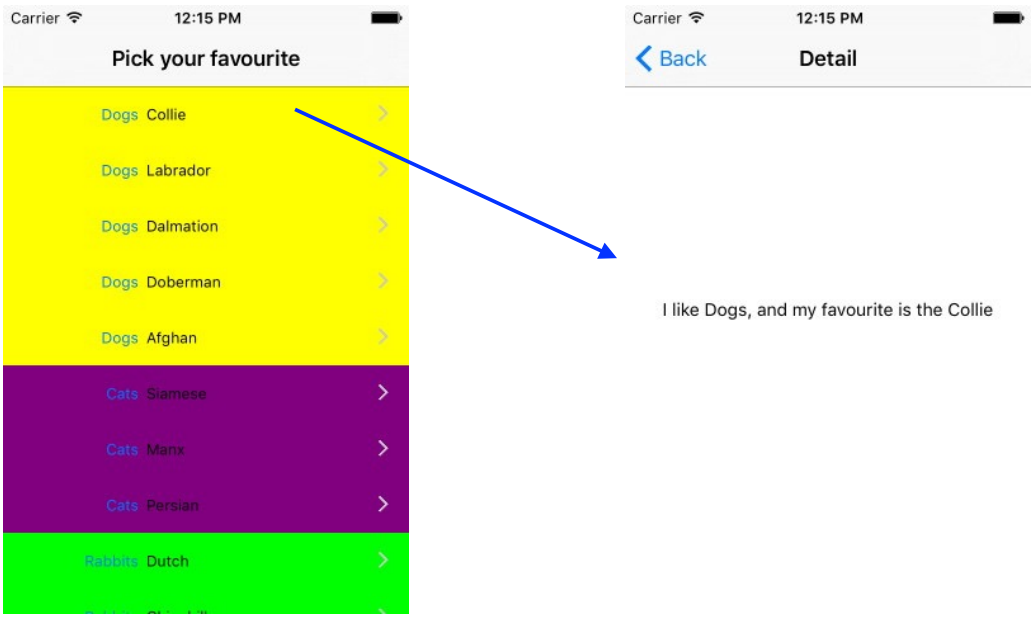


Click here to select the Table

Choose Grouped

Second Animal App
Use structured data and add a detail screen

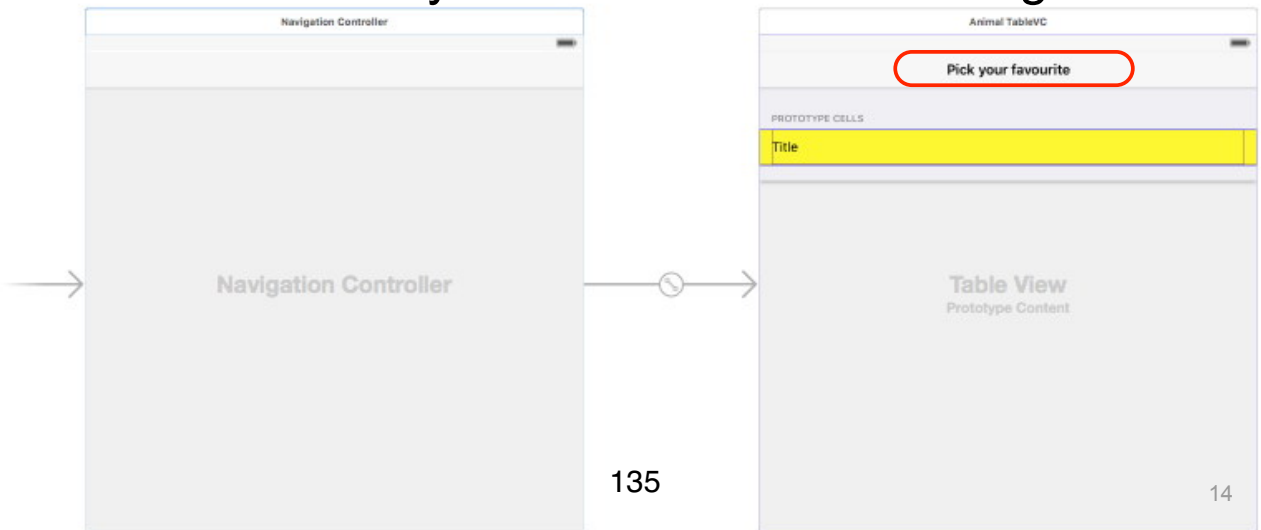
Either use your completed Animal1 App, or use the provided "Animal1 Finished" if you hit problems.



13

Step 1 - Get the structure right

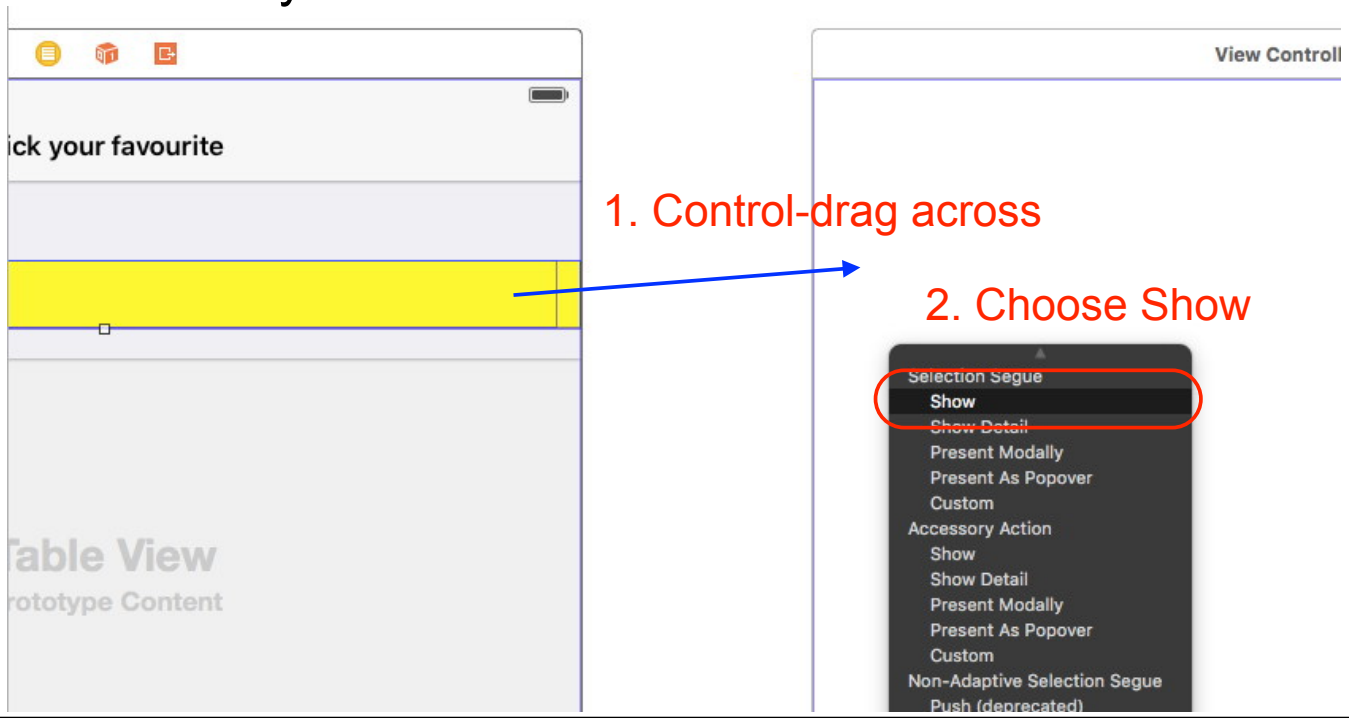
- If we want to push screens then we need a NavigationController - select AnimalTableVC in storyboard, and choose Editor/Embed in/ NavigationController
- Click in the new area at the top of AnimalTableVC, and add the title "Pick your favourite" to the Navigation Item



14

Step 2 - Add the detail screen

- Add another ViewController to the storyboard
- CONTROL-DRAG from the table cell to new controller
- Run and you can move between screens



Step 3 - Add the data

- Add animal data at the top of the class

First line below already exists… add rest underneath it

```
class AnimalTableVC: UITableViewController {

    var categories = ["Dogs", "Cats", "Rabbits" ]

    var categoryItems = [ [ "Collie", "Labrador", "Dalmation",
                            "Doberman", "Afghan"],
                          [ "Siamese", "Manx", "Persian"],
                          [  "Dutch", "Chinchilla", "Lionhead"]]
```

Step 4a - Add basic table code

- Need to say how many sections, and how many in each section - replace previous versions of functions below with these versions

```
override func numberOfSectionsInTableView(tableView:
UITableView) -> Int {
    return categories.count
}

override func tableView(tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
    return categoryItems[section].count
}
```

17

Step 4b - Add basic table code

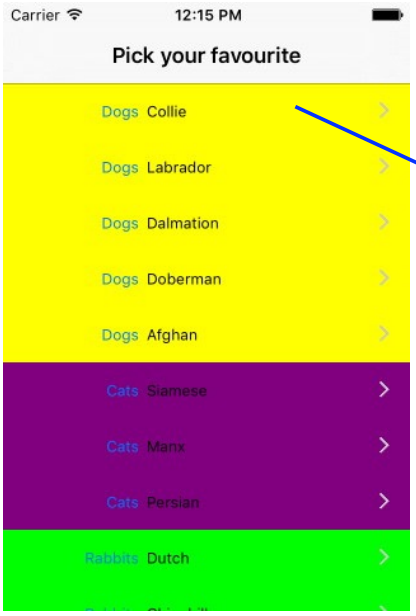
- Need to say what each cell has in it
- Replace existing cellForRowAtIndexPath as below - also need to change cell type in storyboard to be “Left Detail”

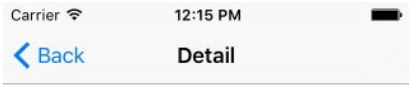
```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "AnimalCell",
                for: indexPath)
    cell.textLabel!.text = categories[indexPath.section]
    cell.detailTextLabel!.text = categoryItems[indexPath.section][indexPath.row]

    switch indexPath.section {
    case 0: cell.backgroundColor = UIColor.yellow
    case 1: cell.backgroundColor = UIColor.purple
    default: cell.backgroundColor = UIColor.green
    }
    return cell
}
```


18

Where are we with Second Animal App?

This screen all works





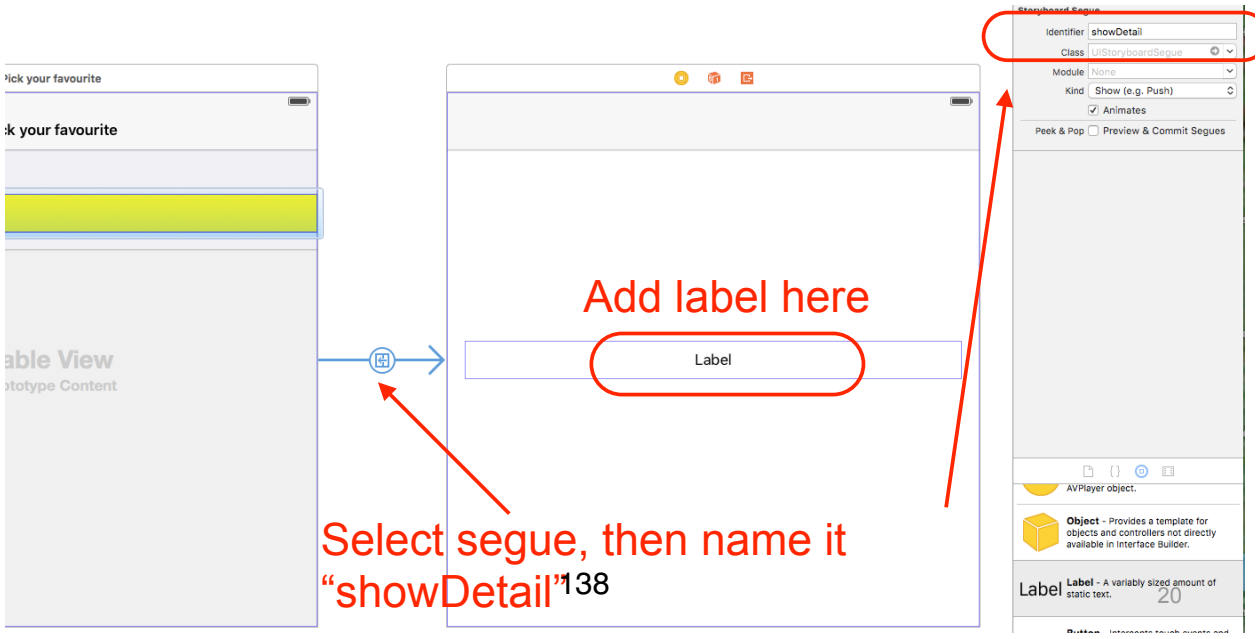
I like Dogs, and my favourite is the Collie

Need to pass the chosen info to the other screen

This happens in "prepareToSegue"

Step 5 - Get the storyboard ready

- Add a label to the detail screen, and name the segue (the transition from one screen to the other)



Step 6a - Need code to set up detail screen

- Create new controller file called AnimalDetailVC.swift as you did for AnimalTableVC.swift (but make a new UIViewController class not a UITableViewController class)
- Associate AnimalDetailVC with the Detail screen by selecting screen in storyboard and choosing it as before in the Identity Inspector
- Use Assistant Editor to drag drop from label on screen to code of AnimalDetailVC to make an @IBOutlet called favourite (as we did in very first app example)
- Run at this point - it should all work, but the label on the detail screen will be blank. We need to add code to pass selected info to the detail screen

Step 7 - Add the code to execute on segue

- In AnimalTableVC.swift, un-comment the declaration of prepareToSegue that is at the end of AnimalTableVC.swift, and add code below

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "showDetail" {
        if let indexPath = self.tableView.indexPathForSelectedRow {
            let detailVC = segue.destination as! AnimalDetailVC
            detailVC.valueForLabel = "I like " + categories[indexPath.section]
                + ", and my favourite is the "
                + categoryItems[indexPath.section][indexPath.row]
        }
    }
}
```

Header exists - add code outlined above

Step 8 - Use the value in AnimalDetailVC

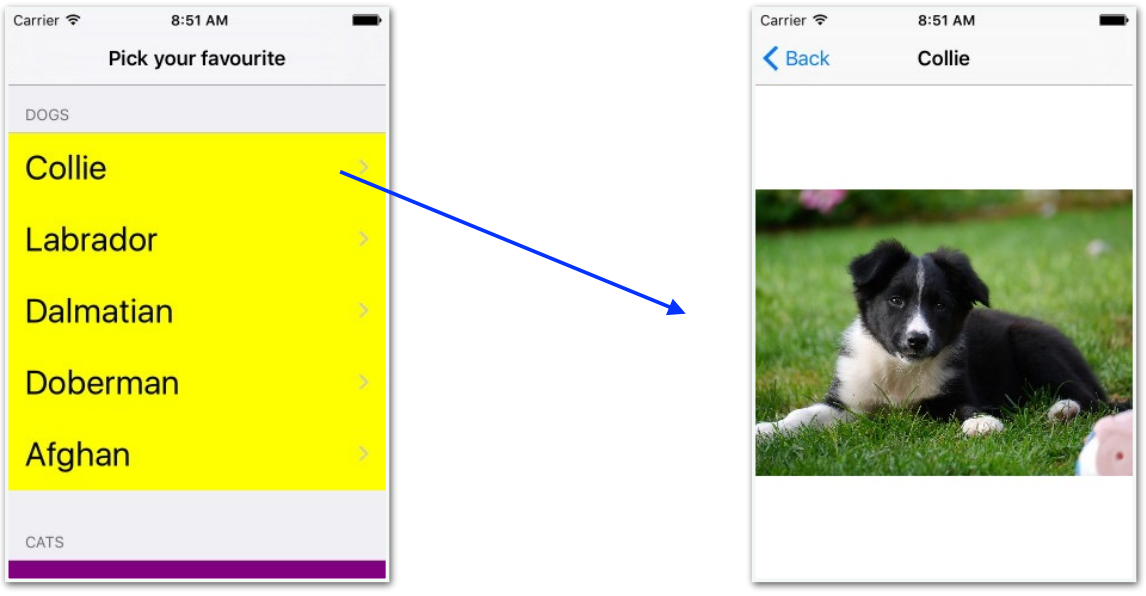
- Could not set the label directly from prepareForSegue as it did not exist at that point. Once the new view controller has been created, can initialise label from passed value

```
class AnimalDetailVC: UIViewController {

    @IBOutlet weak var favourite: UILabel!

    var valueForLabel = ""

    override func viewDidLoad() {
        super.viewDidLoad()

        favourite.text = valueForLabel
    }
```

Add code in red circle instead of existing viewDidLoad

Third Animal App
Put in headers and show photos

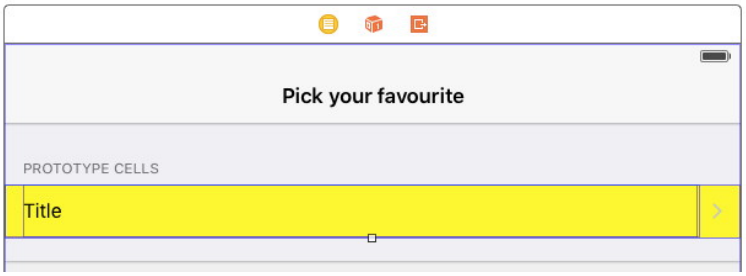
Either use your completed Animal2 App, or use the provided completed version if you hit problems.

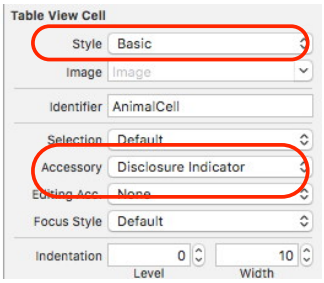


Step 1 - Rearrange AnimalTable

•Change style of AnimalCell to Basic - this will only show a single text. Increase the size of the label text to 28.
•Change Accessory to Disclosure Indicator (this implies there is a lower level of detail available)
•Delete the following line of code from rowAtIndexPath in AnimalTableVC or code will crash:

```
cell.detailTextLabel!.text =
categoryItems[indexPath.section][indexPath.row]
```





25

Step 2 - Add headers and resize

•Add the following routines to AnimalTableVC.swift - the first adds titles for each section, the others resize the height of the section title and the height taken by each cell
•If you run after this, you will see we need another change

```
override func tableView(_  tableView: UITableView, titleForHeaderInSection
section: Int) -> String? {
        return categories[section]
}

override func tableView(_ tableView: UITableView, heightForHeaderInSection
section: Int) -> CGFloat {
        return 40
}

override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {
        return 60
}
```


26

Step 3 - Correct each cell

•Change the line that sets the text in each cell to be what used to be in the subcell

```
cell.textLabel!.text = categoryItems[indexPath.section][indexPath.row]
```

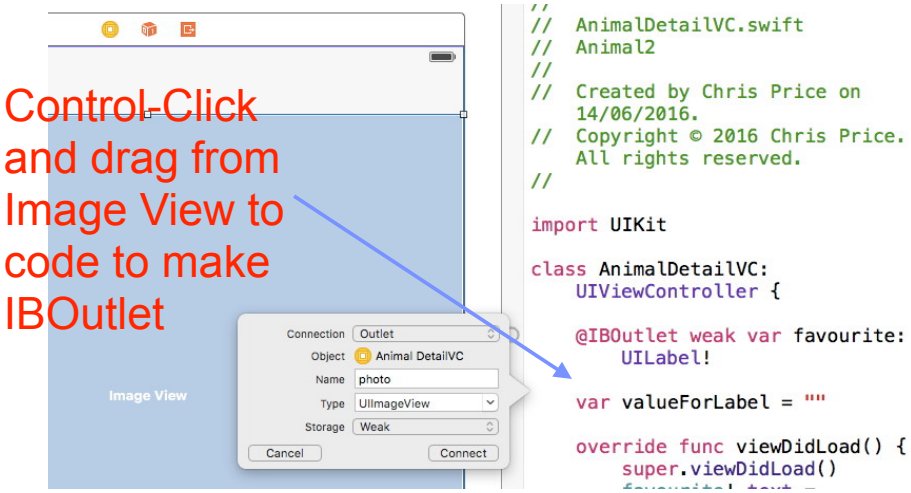
•In prepareForSegue, also change the item that is passed to just be the breed name instead of the long message, and add a title for the screen

```
detailVC.valueForLabel = categoryItems[indexPath.section][indexPath.row]
detailVC.title = categoryItems[indexPath.section][indexPath.row]
```

•If you run now, the table should list breeds correctly, and the detail screen should just say the breed

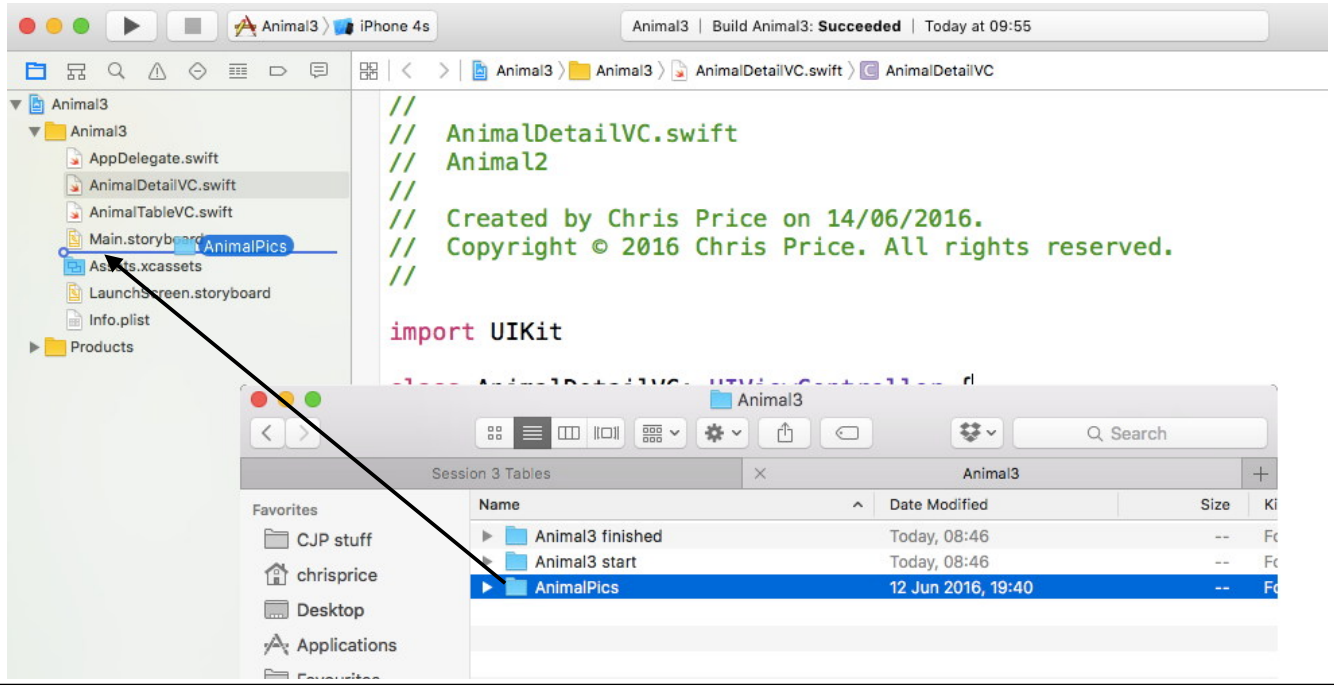
Step 4 - Make detail screen show a picture

•Delete the label from the detail screen in storyboard
•Add an Image View instead
•Add IBOutlet for the Image View to AnimalDetailVC.swift and call it photo



Step 5 - Add all the photos we need

•There is a folder of animal pictures in Animal3 called AnimalPics -drag the folder to the Assets in Xcode, and drop it in - the folder will then be included in the project



Step 6 - Change set up for screen

•Change viewDidLoad in AnimalDetailVC as follows (this will now load the correct photo in the Image View when the screen loads):

```
override func viewDidLoad() {
   super.viewDidLoad()
   self.title = valueForLabel
   photo.image = UIImage(named: "\(valueForLabel).jpg")
}
```

• Tidying up - also delete the now unused outlet below

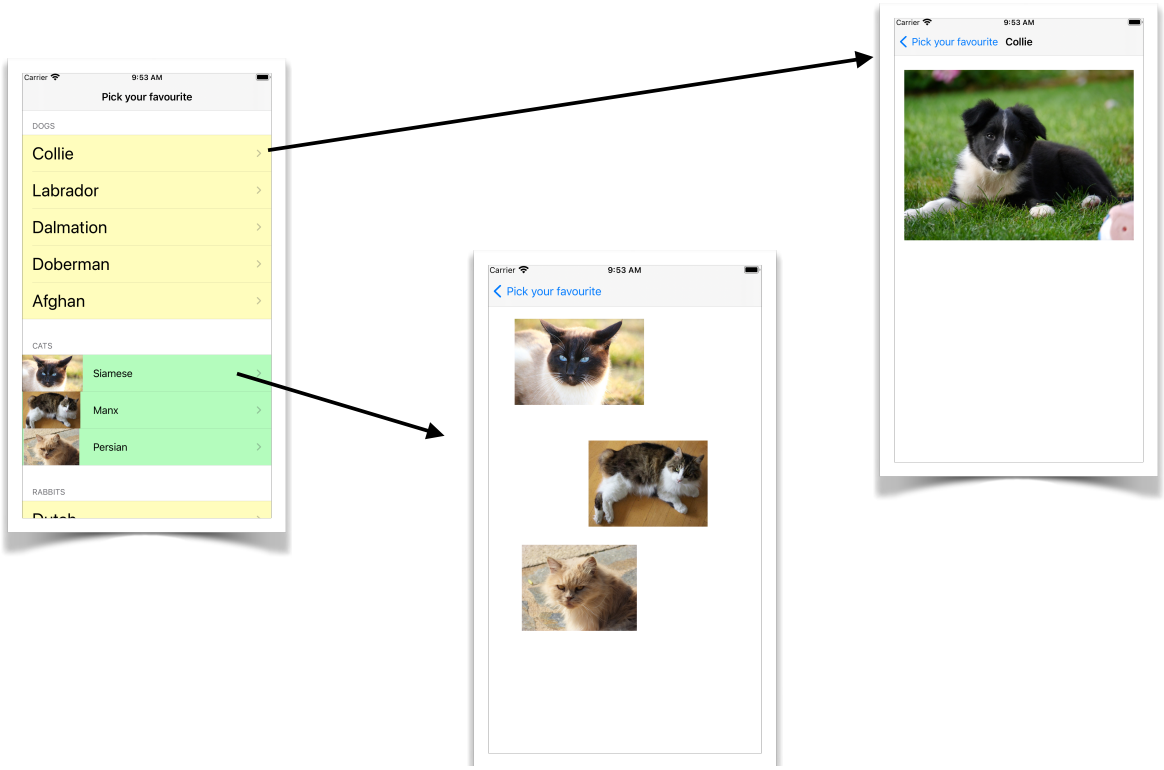
```
@IBOutlet weak var favourite: UILabel!
```

• Finally change the View Mode for the Image View from Scale To Fill to Aspect Fill

Finished!

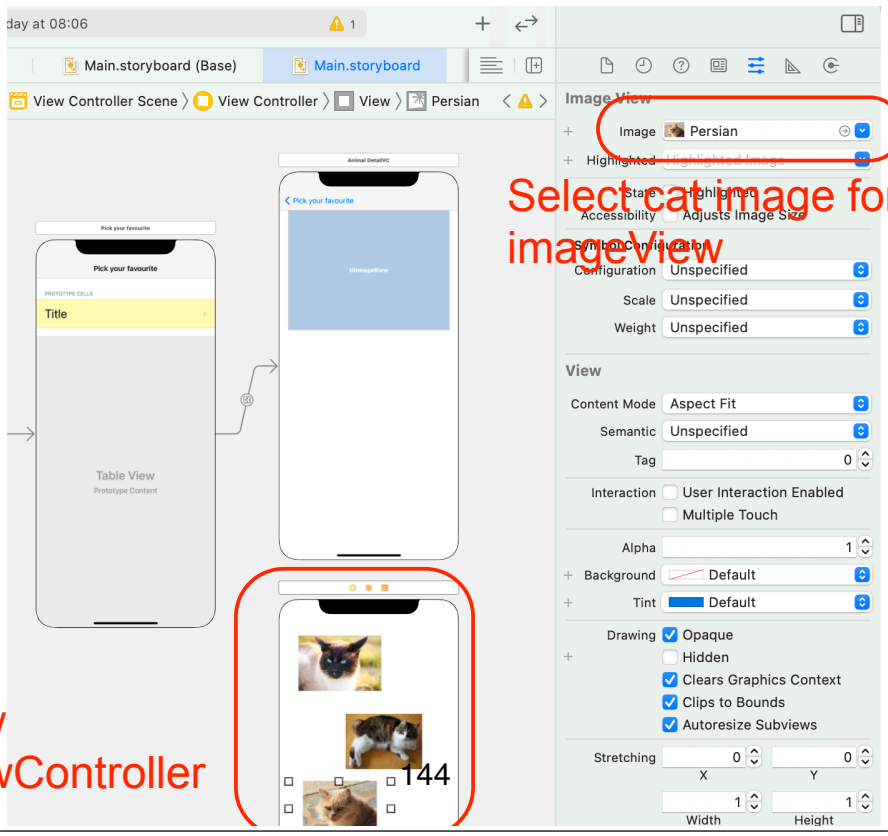
Fourth Animal App
Add custom cells and multiple destinations

Either use your completed Animal3 App, or use the provided completed Animal3 app if you hit problems.



Step 1 - Add a second destination

- Add another ViewController screen to the story board



Select cat image for each imageView

New ViewController

144

Step 2 - Make a second cell type

- Select the Tableview in the TableViewController in the storyboard
- Change the number of prototype cells from 1 to 2
- Make the reuse identifier for the second cell “PictureCell”
- Control drag from the new cell to the new ViewController and choose Show
- Colour the new cell green to be able to tell the difference

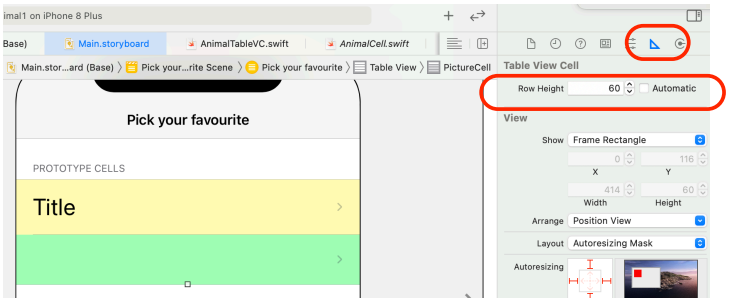
Step 3 - Make cats use new cell

- Select the Tableview in the TableViewController in the storyboard
- Change the number of prototype cells from 1 to 2
- Make the reuse identifier for the second cell “PictureCell”
- Control drag from the new cell to the new ViewController and choose Show
- Colour the new cell green to be able to tell the difference
- Update cellForRowAt indexPath as follows

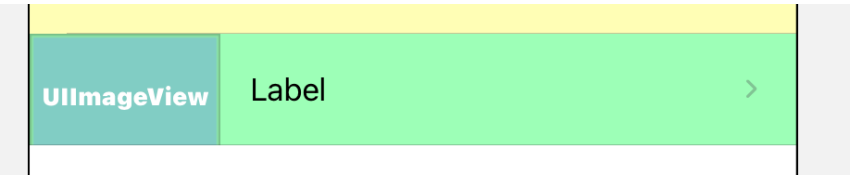
```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "AnimalCell",
                                             for: indexPath)
    // Configure the cell...
    cell.textLabel?.text = "Rabbit, section \(indexPath.section), row \(indexPath.row)"
    return cell
}
```

Step 4 - Make PictureCell a custom cell

•Select the PictureCell in the TableViewController screen in the storyboard
•Change its style to custom and it will go blank
•In the size inspector, make its height to be 60



•Add an image view and a label to the cell



35

Step 4 - Make code for custom PictureCell

•To have a custom Cell, you need to have some definitions and possibly code to go with it. You do this by creating a UITableCell class associated with the cell, and then making IBOutlets in that class for the custom labels and views

•Create a new file (like when you were making view controller code, but this time make it a UITableViewCell class, and call it PictureCell.swift.

•Associate it with the PictureCell table cell in the table by selecting the cell, and then clicking on the identity inspector, and associating the PictureCell class with the cell.

•Add IBOutlets for the Image View (photo) and the Label (animalName) in the table cell in PictureCell.swift

•We can now refer to these outlets in cellForRowAtIndexPath

36

Step 5 - Add code to initialise cells

- Make your cellForRowAtIndexPath look like this:

```
override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
    if indexPath.section == 1 { //Cats
        let cell = tableView.dequeueReusableCell(
            withIdentifier: "PictureCell", for: indexPath) as! PictureCell
        let wantedName =
            categoryItems[indexPath.section][indexPath.row]
        cell.photo.image = UIImage(named: "\(wantedName).jpg")
        cell.animalName.text = wantedName
        return cell
    } else {
        let cell = tableView.dequeueReusableCell(
            withIdentifier: "AnimalCell", for: indexPath)
        cell.textLabel!.text =
            categoryItems[indexPath.section][indexPath.row]
        return cell
    }
}
```

Finished!

37

What we've learned

- How to make dynamic tables
- How to link several views and pass data
- The different kinds of tables and cells that are available and how to use them
- Some of the wide variety of ways in which tables can be used and cells can be configured, but there are many more

38

Trying this for ourselves

- You are ready to start filling in some of the Conference app
- Folder Conference in this session has a set of files with data about the conference and how it can be used
- Try building the Speaker list and individual speaker page from our prototype a few days ago

39

Session 12: Protocols and closures

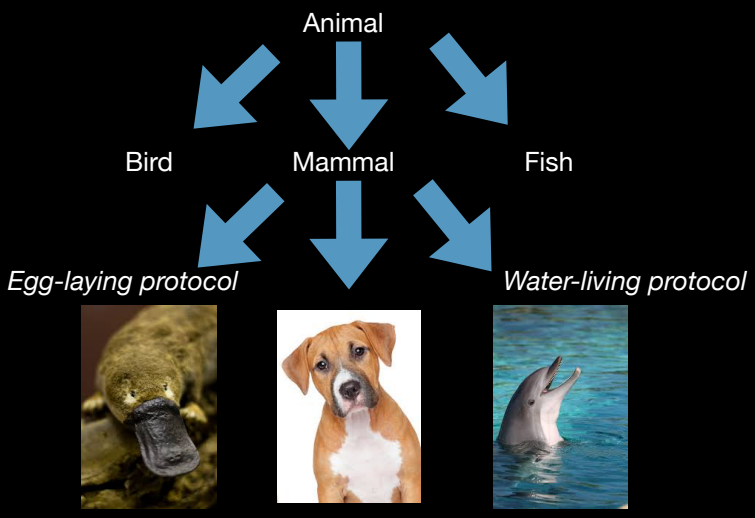
- Built in protocols – CustomStringConvertible, Equatable, Comparable, Codable
- Defining and implementing your own protocols
- Using closures
- Shortening closures
- Lab 12: Protocols and Closures
- This covers lessons 1.1 and 2.1 of Development in Swift Data Collections (the second Apple book)

1

Protocols

What are protocols for?

Shared behaviour that is not necessarily inherited
(And inheritance is not a choice for Structs)



Protocols

Define a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality

Swift standard library defines many protocols, including these:

`CustomStringConvertible`

`Equatable`

`Comparable`

`Codable`

When you adopt a protocol, you must implement all required methods.

Printing with CustomStringConvertible

```
let string = "Hello, world!"
print(string)

let number = 42
print(number)

let boolean = false
print(boolean)
```

```
Hello, world!
42
false
```

Printing with CustomStringConvertible

```
class Shoe {
  let color: String
  let size: Int
  let hasLaces: Bool

  init(color: String, size: Int, hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }
}

let myShoe = Shoe(color: "Black", size: 12, hasLaces: true)
print(myShoe)
```

```
__lldb_expr_1.Shoe
```

```
class Shoe: CustomStringConvertible {
    let color: String
    let size: Int
    let hasLaces: Bool

    init(color: String, size: Int, hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }









}
```

```
class Shoe: CustomStringConvertible {
    let color: String
    let size: Int
    let hasLaces: Bool

    init(color: String, size: Int, hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }

    var description: String {
        return "Shoe(color: \(color), size: \(size), hasLaces: \(hasLaces))"

    }
}
let myShoe = Shoe(color: "Black", size: 12, hasLaces: true)
print(myShoe)

Shoe(color: Black, size: 12, hasLaces: true)
```

Comparing information with Equatable

```
struct Employee {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String
}

struct Company {
  let name: String
  let employees: [Employee]
}
```

Comparing information with Equatable

```
let currentEmployee = Session.currentEmployee
let selectedEmployee = Employee(firstName: "Jacob", lastName: "Edwards",
                                jobTitle: "Marketing Director", phoneNumber: "415-555-9293")

if currentEmployee == selectedEmployee {
  // Enable "Edit" button
}
```

Comparing information with Equatable

```
struct Employee: Equatable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
    // Logic that determines if the value on the left hand side and right hand side are equal
  }

}
```

Comparing information with Equatable

```
struct Employee: Equatable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
    return lhs.firstName == rhs.firstName && lhs.lastName == rhs.lastName
  }
}
```

Comparing information with Equatable

```
let currentEmployee = Employee(firstName: "Jacob", lastName: "Edwards",
    jobTitle: "Industrial Designer", phoneNumber: "415-555-7766")
let selectedEmployee = Employee(firstName: "Jacob", lastName: "Edwards",
    jobTitle: "Marketing Director", phoneNumber: "415-555-9293")

if currentEmployee == selectedEmployee {
  // Enable "Edit" button
}
```

Comparing information with Equatable

```
struct Employee: Equatable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
      return lhs.firstName == rhs.firstName && lhs.lastName == rhs.lastName
             && lhs.jobTitle == rhs.jobTitle && lhs.phoneNumber == rhs.phoneNumber
  }
}
```

Sorting information with Comparable

```
let employee1 = Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk",
phoneNumber: "415-555-7767")
let employee2 = Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber:
"415-555-7768")
let employee3 = Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager",
phoneNumber: "415-555-7770")
let employee4 = Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant",
phoneNumber: "415-555-7771")
let employee5 = Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead",
phoneNumber: "415-555-7772")

let employees = [employee1, employee2, employee3, employee4, employee5]
```

```
struct Employee: Equatable, Comparable {
  let firstName: String
  let lastName: String
  let jobTitle: String
  let phoneNumber: String

  static func ==(lhs: Employee, rhs: Employee) -> Bool {
      return lhs.firstName == rhs.firstName && lhs.lastName == rhs.lastName
             && lhs.jobTitle == rhs.jobTitle && lhs.phoneNumber == rhs.phoneNumber
  }

  static func < (lhs: Employee, rhs: Employee) -> Bool {
    return lhs.lastName < rhs.lastName
  }
}
```

```
let employees = [employee1, employee2, employee3, employee4, employee5]

let sortedEmployees = employees.sorted(by:<)

for employee in sortedEmployees {
  print(employee)
}
```

```
Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk", phoneNumber: "415-555-7767")
Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber: "415-555-7768")
Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead", phoneNumber: "415-555-7772")
Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant", phoneNumber: "415-555-7771")
Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager", phoneNumber: "415-555-7770")
```

```
let employees = [employee1, employee2, employee3, employee4, employee5]

let sortedEmployees = employees.sorted(by:>)

for employee in sortedEmployees {
  print(employee)
}
```

```
Employee(firstName: "Grant", lastName: "Phelps", jobTitle: "Senior Manager", phoneNumber: "415-555-7770")
Employee(firstName: "Sang", lastName: "Han", jobTitle: "Accountant", phoneNumber: "415-555-7771")
Employee(firstName: "Daren", lastName: "Estrada", jobTitle: "Sales Lead", phoneNumber: "415-555-7772")
Employee(firstName: "Vera", lastName: "Carr", jobTitle: "CEO", phoneNumber: "415-555-7768")
Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk", phoneNumber: "415-555-7767")
```

Encoding and decoding objects with Codable

```
struct Employee: Equatable, Comparable, Codable {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String

    static func ==(lhs: Employee, rhs: Employee) -> Bool {
        return lhs.firstName == rhs.firstName && lhs.lastName ==
            rhs.lastName && lhs.jobTitle == rhs.jobTitle &&
            lhs.phoneNumber == rhs.phoneNumber
    }

    static func < (lhs: Employee, rhs: Employee) -> Bool {
        return lhs.lastName < rhs.lastName
    }
}
```

Encoding and decoding objects with Codable

```
let ben = Employee(firstName: "Ben", lastName: "Atkins", jobTitle: "Front Desk",
                   phoneNumber: "415-555-7767")

let jsonEncoder = JSONEncoder()
if let jsonData = try? jsonEncoder.encode(ben),
    let jsonString = String(data: jsonData, encoding: .utf8) {
    print(jsonString)
}
```

```
{"firstName":"Ben","lastName":"Atkins","jobTitle":"Front Desk","phoneNumber":"415-555-7767"}
```

Creating a protocol

```
protocol FullyNamed {
  var fullName: String { get }

  func sayFullName()
}

struct Person: FullyNamed {
  var firstName: String
  var lastName: String
}
```

Creating a protocol

```
struct Person: FullyNamed {
  var firstName: String
  var lastName: String

  var fullName: String {
    return "\(firstName) \(lastName)"
  }

  func sayFullName() {
    print(fullName)
  }
}
```

Lab: Protocols

Open and complete the first page of exercises in `Lab - Protocols.playground`

Closures

Closures

```
(firstTrack: Track, secondTrack: Track) -> Bool in
 return firstTrack.trackNumber < secondTrack.trackNumber
```

```
let sortedTracks = tracks.sorted (    )
```

Syntax

```
func sum(numbers: [Int]) -> Int {
  // Code that adds together the numbers array
  return total
}
```

```
let sumClosure = { (numbers: [Int]) -> Int in
  // Code that adds together the numbers array
  return total
}
```

```
// A closure with no parameters and no return value
let printClosure = { () -> Void in
  print("This closure does not take any parameters and does not return a value.")
}

// A closure with parameters and no return value
let printClosure = { (string: String) -> Void in
  print(string)
}

// A closure with no parameters and a return value
let randomNumberClosure = { () -> Int in
  // Code that returns a random number
}

// A closure with parameters and a return value
let randomNumberClosure = { (minValue: Int, maxValue: Int) -> Int in
  // Code that returns a random number between `minValue` and `maxValue`
}
```

Passing closures as arguments

```
let sortedTracks = tracks.sorted { (firstTrack: Track, secondTrack: Track) -> Bool in
  return firstTrack.trackNumber < secondTrack.trackNumber
}
```

```
 let sortedTracks = tracks.sorted { (firstTrack: Track, secondTrack: Track) -> Bool in
   return firstTrack.starRating < secondTrack.starRating
 }
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack: Track, secondTrack: Track) -> Bool in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) -> Bool in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) in
  return firstTrack.starRating < secondTrack.starRating
}
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { return $0.starRating < $1.starRating }
```

Syntactic sugar

```
let sortedTracks = tracks.sorted { $0.starRating < $1.starRating }
```

Syntactic sugar

```
let sortedTracks = tracks.sorted(by: <)
```

Collection functions using closures

Map

Filter

Reduce

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates an empty array that will be used
// to store the full names
var fullNames: [String] = []

for name in firstNames {
    let fullName = name + " Smith"
    fullNames.append(fullName)
}
```

fullNames

0	"Johnny Smith"
1	"Nellie Smith"
2	"Aaron Smith"
3	"Rachel Smith"

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates a new array of full names by adding "Smith"
// to each first name
let fullNames = firstNames.map { (name) -> String in
    return name + " Smith"
}
```

fullNames

0	"Johnny Smith"
1	"Nellie Smith"
2	"Aaron Smith"
3	"Rachel Smith"

Collection functions using closures

```
// Initial array
let firstNames = ["Johnny", "Nellie", "Aaron", "Rachel"]

// Creates a new array of full names by adding "Smith"
// to each first name
let fullNames = firstNames.map { $0 + " Smith" }
```

fullNames

0	"Johnny Smith"
1	"Nellie Smith"
2	"Aaron Smith"
3	"Rachel Smith"

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

var numbersLessThan20: [Int] = []

for number in numbers {
    if number < 20 {
        numbersLessThan20.append(number)
    }
}
```

numbersLessThan20

0	4
1	8
2	15
3	16

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

let numbersLessThan20 = numbers.filter { (number) -> Bool in

    return number < 20
}
```

numbersLessThan20

0	4
1	8
2	15
3	16

Collection functions using closures

```
let numbers = [4, 8, 15, 16, 23, 42]

let numbersLessThan20 = numbers.filter { $0 < 20 }
```

numbersLessThan20

0	4
1	8
2	15
3	16

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

var total = 0

for number in numbers {
    total = total + number
}
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0) { (currentTotal, newValue) -> Int in
    return currentTotal + newValue
}
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0, {$0 + $1})
```

Collection functions using closures

```
let numbers = [8, 6, 7, 5, 3, 0, 9]

let total = numbers.reduce(0, +)
```

Closures capture their environment

```
animate {
  self.view.backgroundColor = .red
}
```

Lab: Closures

Open and complete the exercises in Lab - Closures.playground

Session 13: Mapping and Location

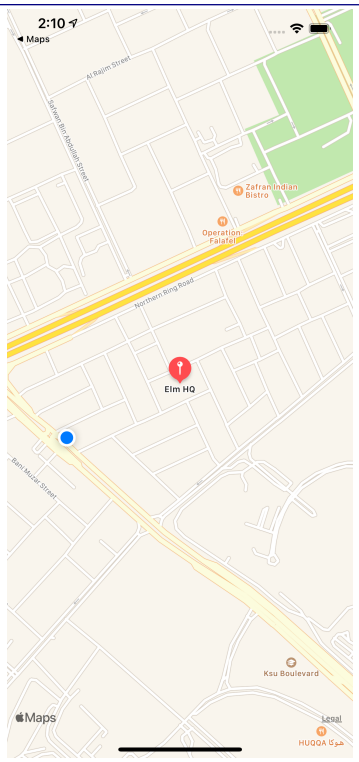
- Setting up location services
- Tracking movement
- Annotating maps
- Lab 13: Adding Maps to an app

- This material is not covered in the Apple Books

1

What we are going to build

- A mapping screen which shows landmarks plus our own position



2

Step 1

•Start a new simple app
•Add a map view to the screen - constrain it to take up the whole window
•Make an outlet for the map View in the associated file (ViewController.swift) and call it *mapView*

The program will fail to compile at this time, as it does not know what an MKMapView is - you can fix this by adding:

Import MapKit

at the top of the screen

If you run it at this point, your app will show a map centred on the country - we need to make it more focused on the place we want to show

Step 2

•We need to define a class that supports the MKAnnotation protocol - this has the right info to define a labelled point on the map

```
class MapNode: NSObject, MKAnnotation {
    let coordinate: CLLocationCoordinate2D
    let title: String?
    let subtitle: String?

    init(latitude: CLLocationDegrees, longitude: CLLocationDegrees, title: String,
subtitle: String) {
        coordinate = CLLocationCoordinate2D(latitude: latitude, longitude: longitude)
        self.title = title
        self.subtitle = subtitle
    }
}
```

Step 3

•In viewDidLoad, we will define an annotation for Elm HQ

```
let elmHQ = MapNode(latitude: 24.747842, longitude: 46.623180, title: "Elm HQ",
subtitle: "Headquarters of Elm Information Security Company")
```

•Also in viewDidLoad, we will set the map to centre on ElmHQ, and add the annotation to the map

```
//Set a coordinate region with ElmHQ as centre and a kilometer span
let regionSpan: CLLocationDistance = 1000
let coordinateRegion = MKCoordinateRegion( center: elmHQ.coordinate,
                    latitudinalMeters: regionSpan,
                    longitudinalMeters: regionSpan)
mapView.setRegion(coordinateRegion, animated: true)
mapView.addAnnotation(elmHQ)
```

•If we run the app now, our annotation is on the map - now we want to know where WE are on the map

We want to show OUR location

- Four things needed for that
  - The view controller needs to conform to CLLocationManagerDelegate
  - You need a CLLocationManager set up
  - You need to tell it to show the user location on the map
  - You need to set it so the app asks for permission to monitor user position

Step 4

•At the start of the ViewController class, state it uses the CLLocationManagerDelegate, and declare a Manager

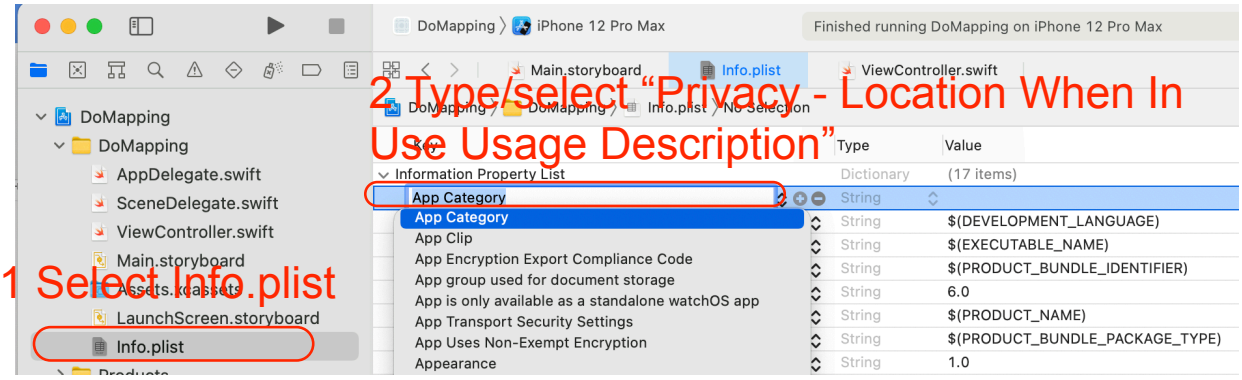
```
class ViewController: UIViewController, CLLocationManagerDelegate {

    let locationManager = CLLocationManager()
```

•At the end of viewDidLoad, add code to ask for permission to use the user location, and give it a reasonable accuracy, then show user location

```
locationManager.requestWhenInUseAuthorization()
locationManager.distanceFilter = kCLDistanceFilterNone
locationManager.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
locationManager.startUpdatingLocation()
mapView.showsUserLocation = true
```


Step 5

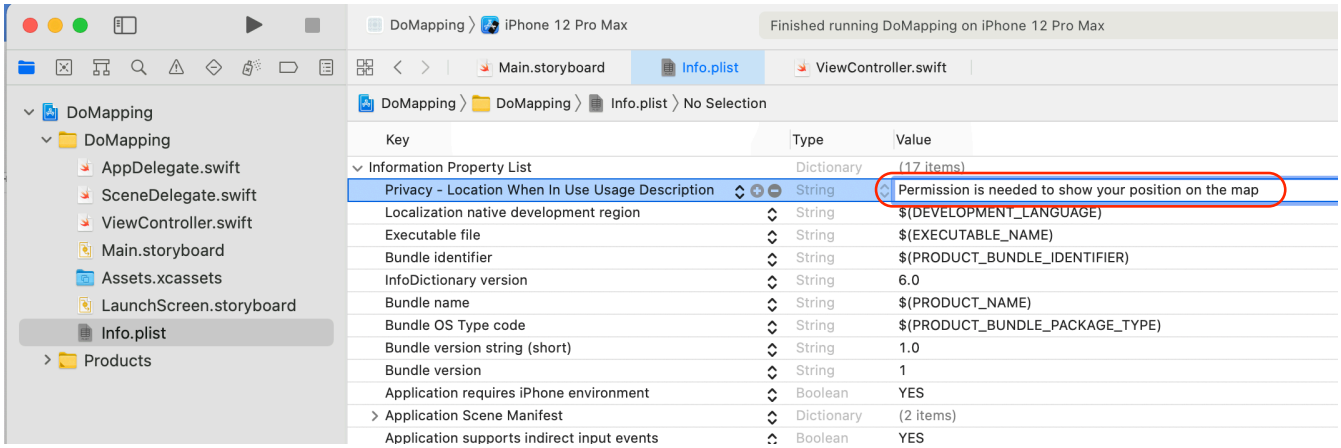
•We now have all the code we need, but when you are doing something you need permission for (using Location, accessing the camera, sending Notifications), you also need to signal that permission in the app's info.plist file



•Click on the + next to Information Property List, and you will get an extra line as shown above. Start typing "Privacy - Location When In Use Usage Description" and it should appear and can be selected

Step 6

•Add a value for the Privacy prompt - this will be used to explain to the user why you want access



Step 7

•The app should now run fine. It will prompt for our position, and on a real device, it will show as a blue dot
•On the simulator, we need to provide a simulated location for where we are on the map
•Lets say we are at the Doka Bakery a few streets away

•In the Simulator, choose Features/Location/ Custom Location
•Put in values 24.746383, 46.619839

Now we should show on the map as a blue dot

Applying to case study

- You should now be able to add the map screen to the Conference app - either to show a specific selected location or to have a map that shows all the locations (you can add a list of landmarks instead of just one).

Session 14: Gestures and accelerometer

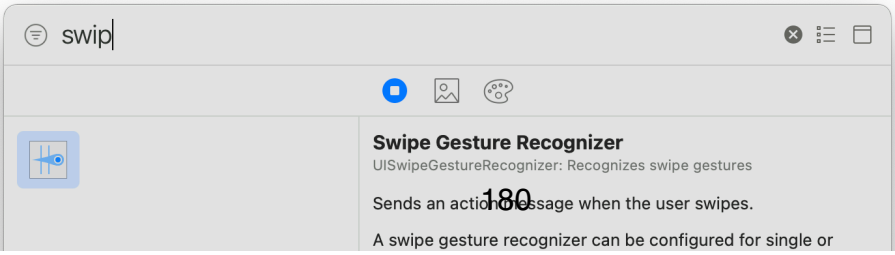
- Implementing taps and swipes
- Using the accelerometer
- Lab 14: Example apps using taps, swipes and the accelerometer

- This material is not well covered in the Apple Books - there is a gesture example on page 280-281 of Develop in Swift Fundamentals

1

Implementing swipes and taps

- **Step 1**
- Add a label to the screen for the ViewController and give it the value zero.
- Add a swipe gesture recognizer from the library to the screen on the main storyboard.



2

Implementing swipes and taps

- **Step 2**
- Make label outlet called *counter*
- Make swipe action outlet for swipe gesture called rightSwipe
- Add second Swipe gesture recognizer to the main screen, and set its attribute to be a left swiper, and add a swipe action outlet called leftSwipe

3

Implementing swipes and taps

- **Step 3**
- Keep a count of the value of counter, and when there is a right swipe increase it if it is less than 9
- When there is a left swipe, decrease it if it is greater than 0
- Update the counter to reflect value change

4

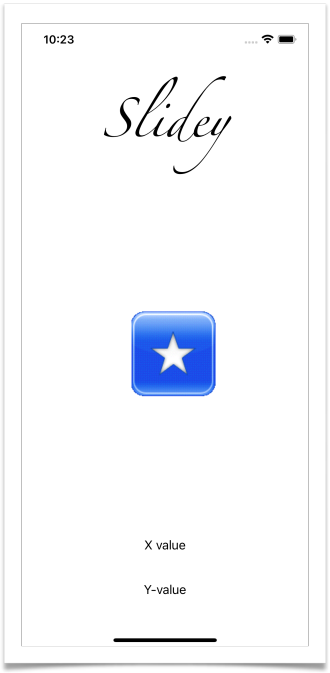
Exploring gestures further

- Try adding a double tap recogniser by picking a tap recogniser and setting number of taps to 2
- Action the double tap to clear the counter back to zero whatever its value

5

Using the accelerometer

- Project SlideySwift shows how phone orientation information can be used as input
- This will only work on actual phones, as no simulated accelerometer data



6

Need to set up monitoring

```
override func viewDidLoad() {
    super.viewDidLoad()
    if (motionManager.isDeviceMotionAvailable) {

       // Update 60 times per second
       motionManager.deviceMotionUpdateInterval = 1.0/60.0

       let timer = Timer(fire: Date(), interval: (1.0/60.0),
                   repeats: true, block: { (timer) in
                     // Get the accelerometer data.
                            self.accelUpdate()
                     })
       RunLoop.current.add(timer, forMode: .default)
   } else {
       print("Can't use CMMotion on this device");
   }
}
```

7

Turn on only when screen showing

```
override func viewDidAppear(_ animated: Bool) {
    motionManager.startAccelerometerUpdates()
}

override func viewDidDisappear(_ animated: Bool) {
   motionManager.stopAccelerometerUpdates()
}
```


8

What happens in timer

```
func accelUpdate() {
    if let accelerometerData = motionManager.accelerometerData {
       let xGravity = accelerometerData.acceleration.x
       let yGravity = accelerometerData.acceleration.y
       // Write values on screen
       xValue.text = String(format: "x accel is %.1f", xGravity)
       yValue.text = String(format: "y accel is %.1f", yGravity)
       let xChange = (xGravity*40)
       let yChange = -(yGravity*40)
       // Calculate values and move unless against edge
       if ((self.slidey.center.x > 75) && (xChange < 0))
          || ((self.slidey.center.x < self.view.frame.size.width-75)
          && (xChange > 0)) {
         self.slidey.center.x = CGFloat(Double(self.slidey.center.x)+xChange)
       }
       if ((self.slidey.center.y > 75) && (yChange < 0))
          || ((self.slidey.center.y < self.view.frame.size.height-75)
          && (yChange > 0)) {
         self.slidey.center.y = CGFloat(Double(self.slidey.center.y)+yChange)
       }
    }
}
```


Things to note

- Detecting swipes / accelerometer / location detection are all heavy on the battery
- Turn them off when you don’t need them
- Less frequent or less accurate monitoring is less heavy