framework training
We love technology

Building iOS Apps in Swift

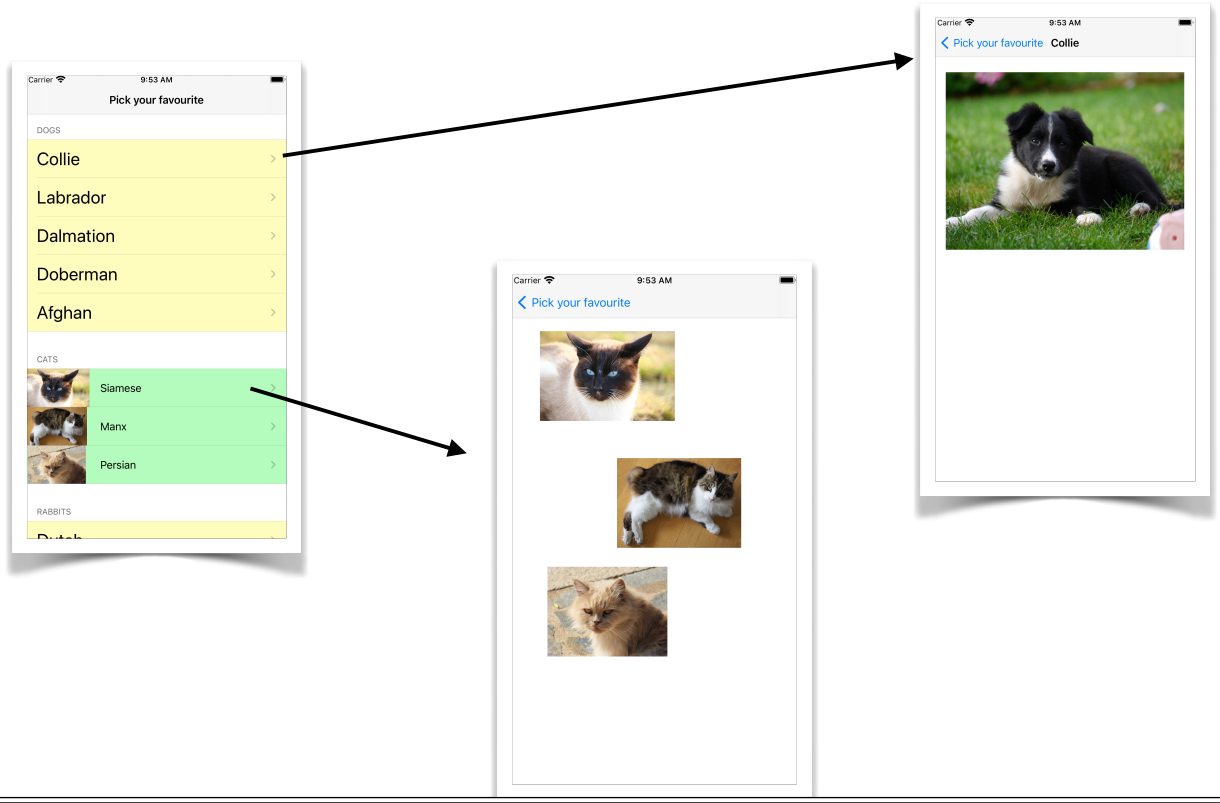
Course notes for part 2

May 2022

020 3137 3920 @FrameworkTrain

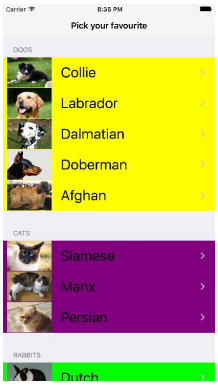
frameworktraining.co.uk

Session 11: 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.

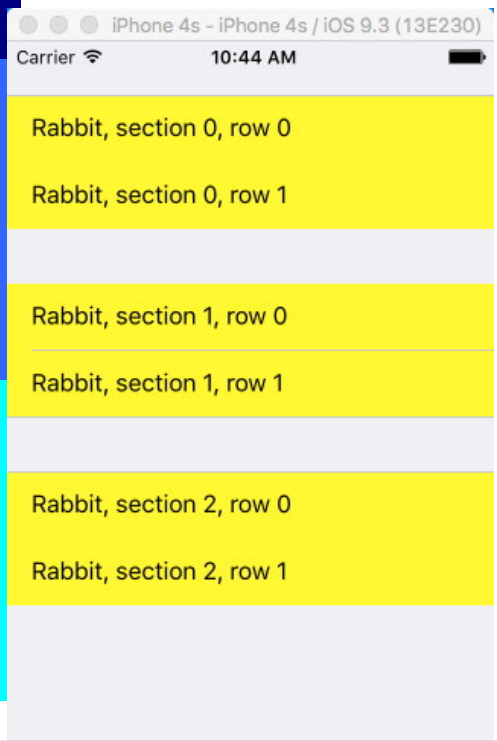
There are two sets of delegates associated with a table:

- 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

First Animal App - Minimum answers for a table

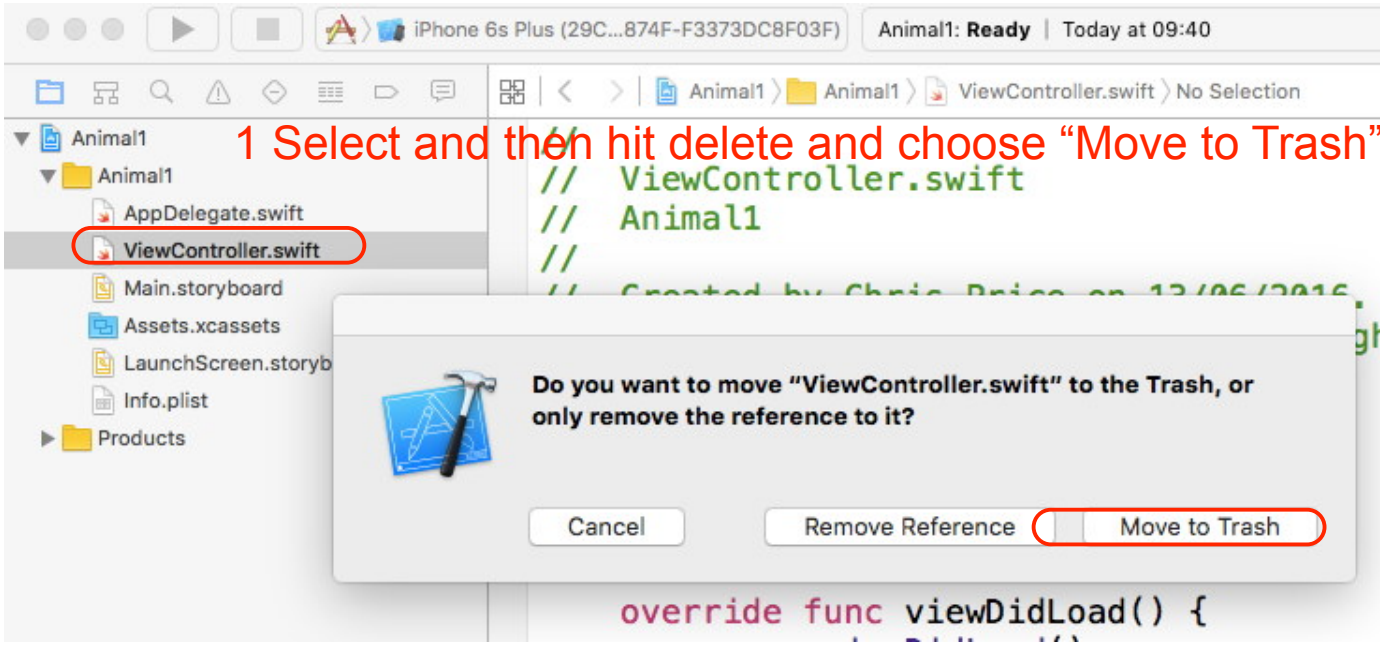


Step 1

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

Step 2

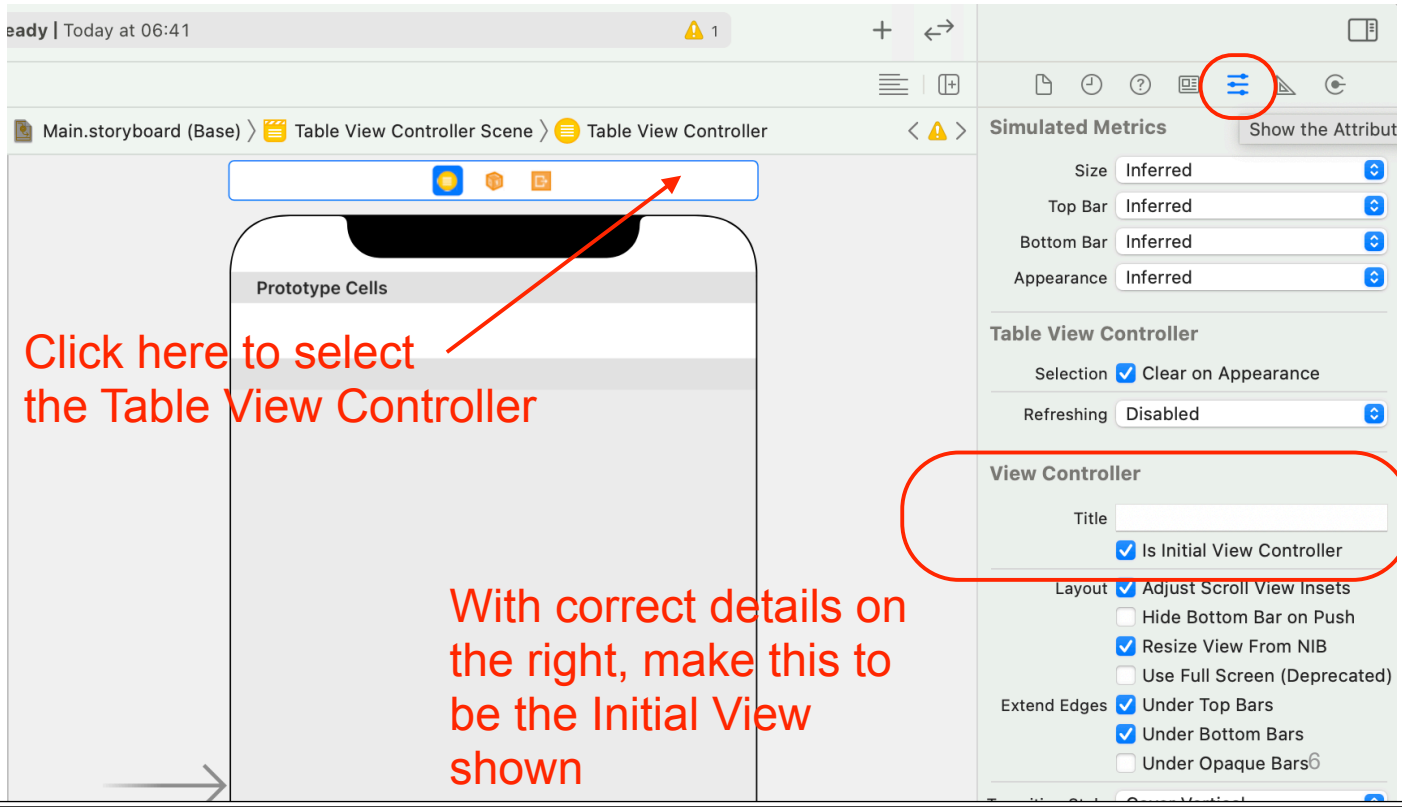
- Delete ViewController.swift as we won't be using it.
- Go to Main.storyboard, select the window and delete it (as we did when making static tables).



2 Select storyboard and get rid of the ViewController view

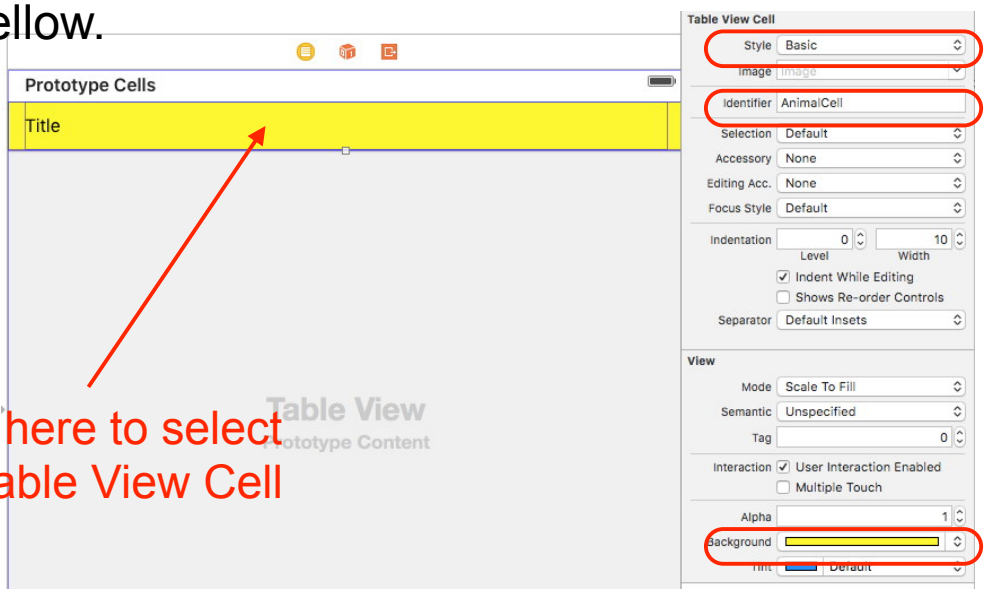
Step 3

- 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.



Style Identifier

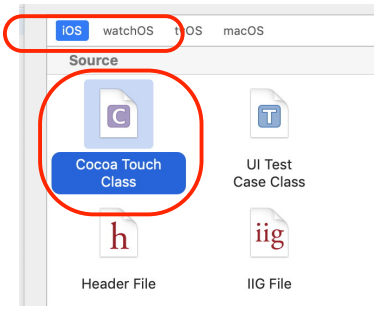
Click here to select the Table View Cell

Background

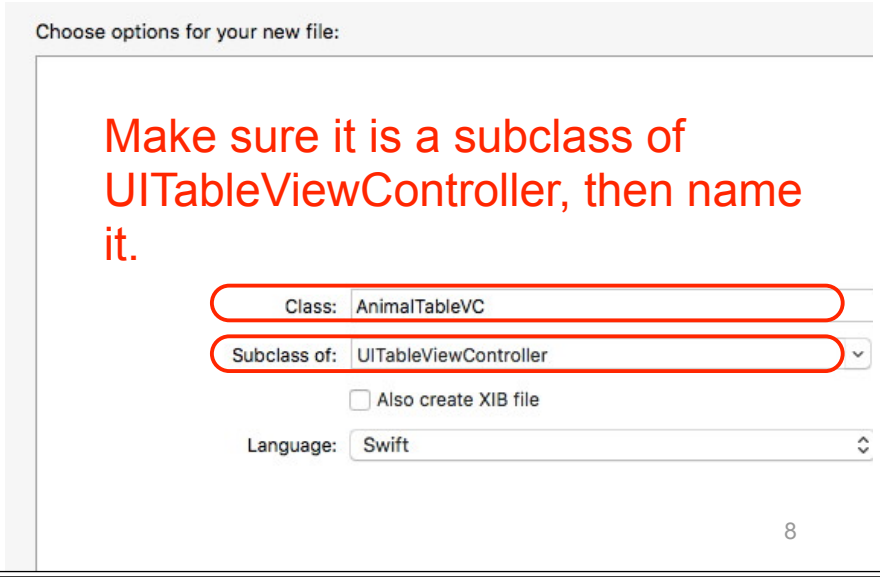
- You can run your app now, but it will look pretty dull as you don't have any code associated with the View.
- You now need to create a UITableViewController file and write some code to define what goes in the table.

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



Make sure it is a subclass of UITableViewController, then name it.

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.

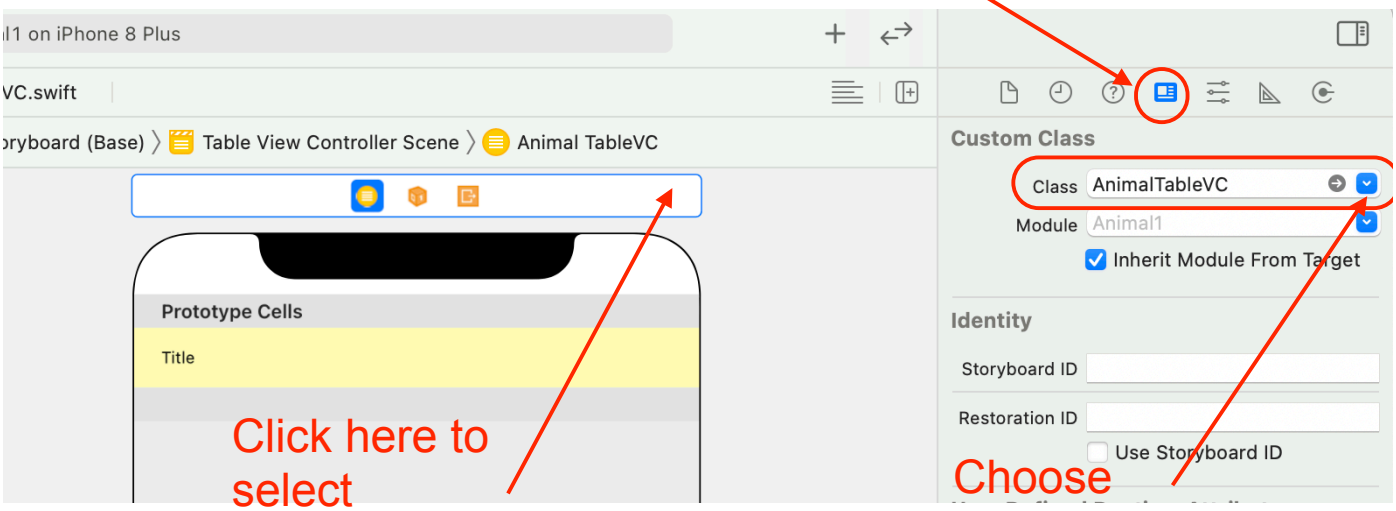
Make cellForRowAtIndexPath look like this:

```
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.



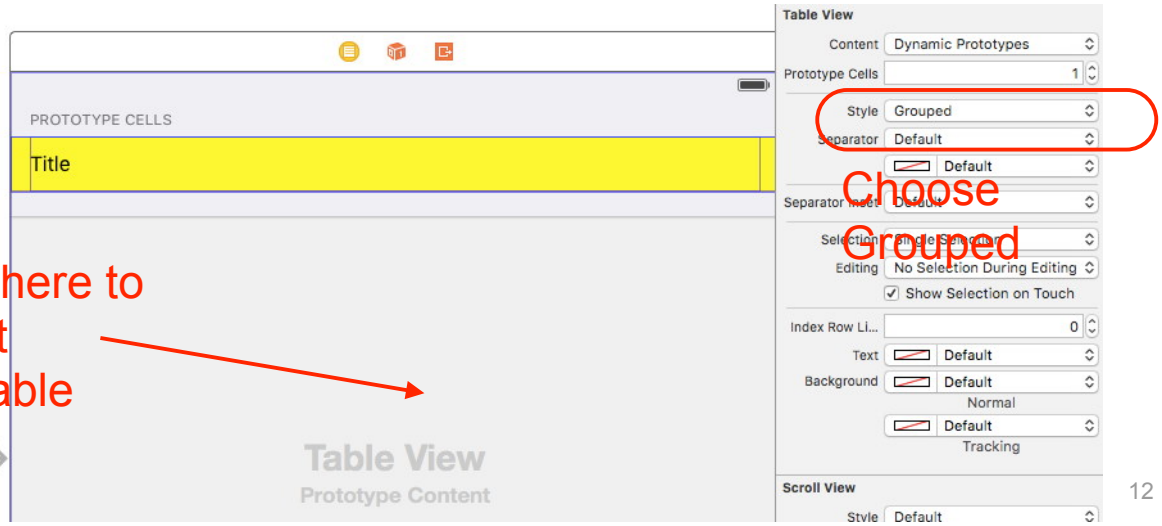
Click here to select the TableView Controller screen

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.

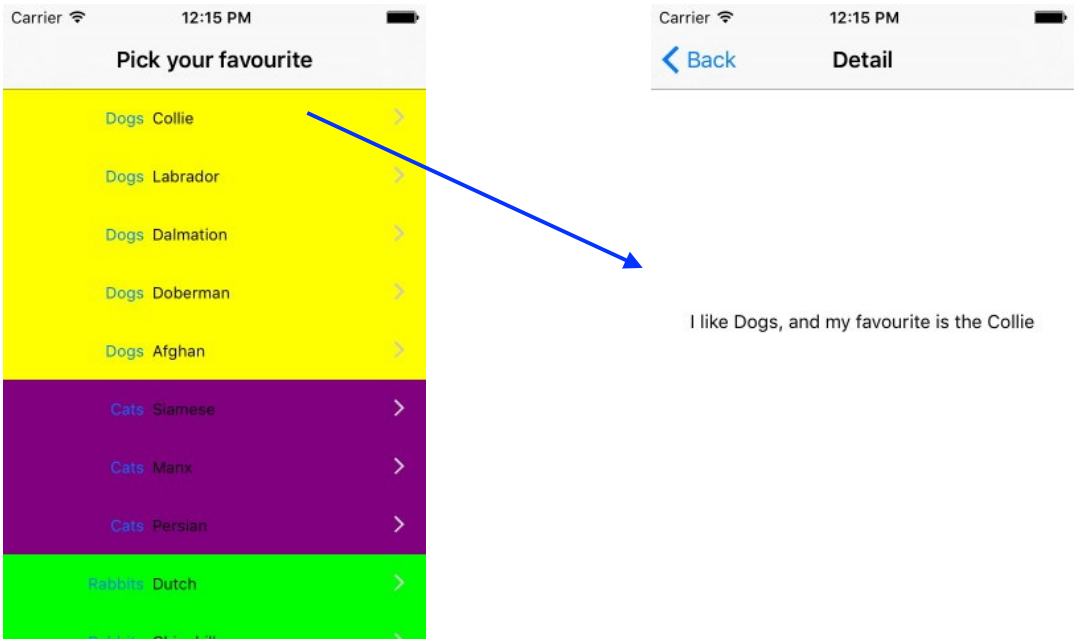


Click here to select the Table

Choose Grouped

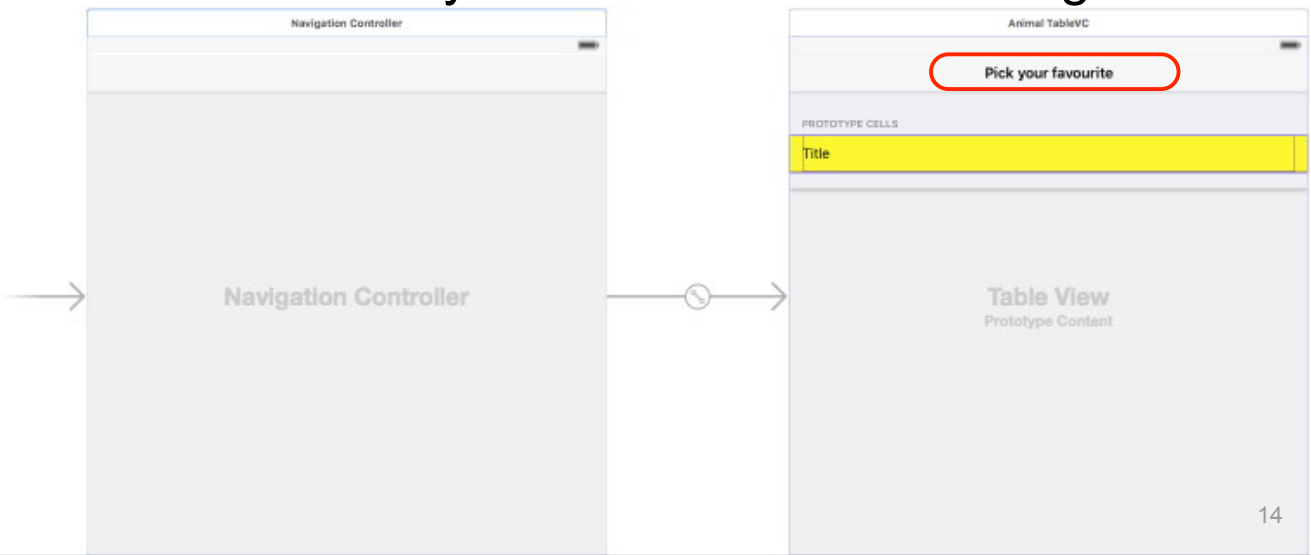
Second Animal App
Use structured data and add a detail screen

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



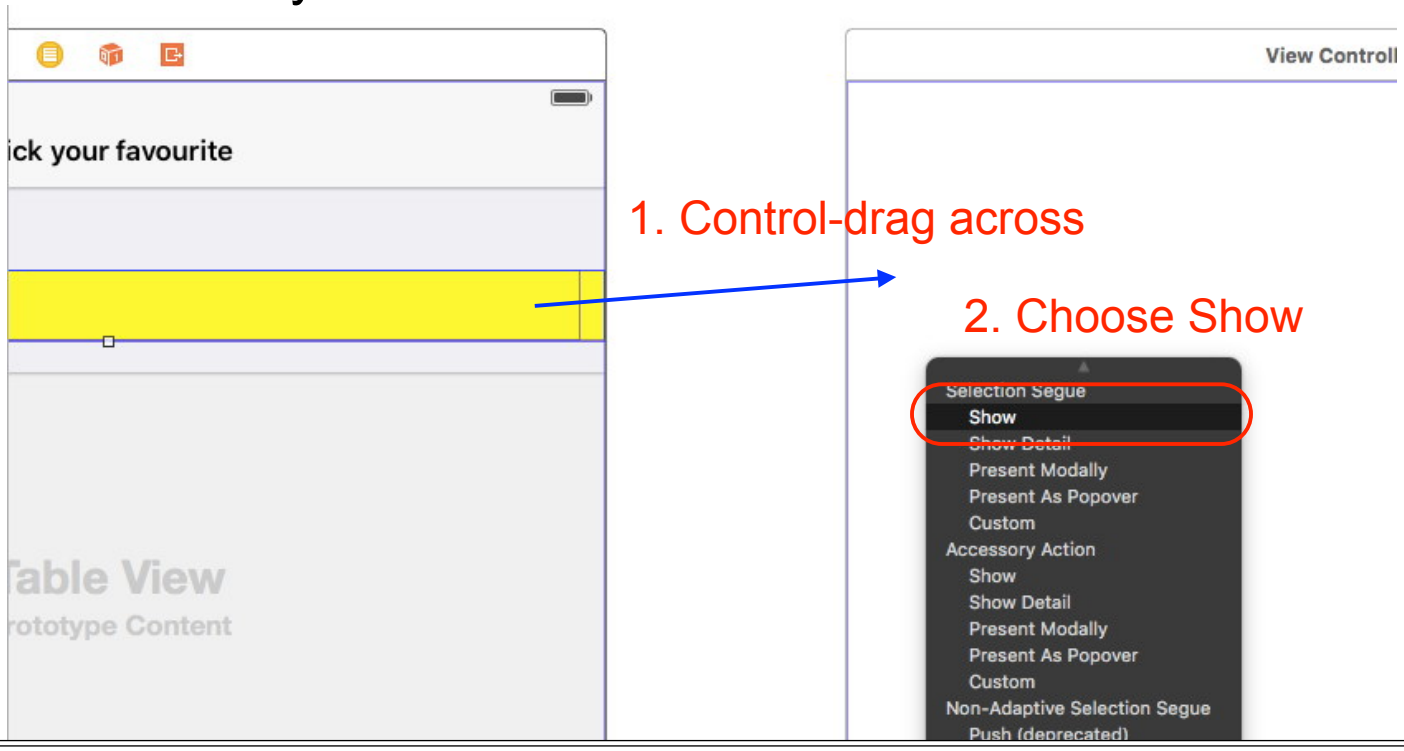
Step 1 - Get the structure right

- 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



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
}
```

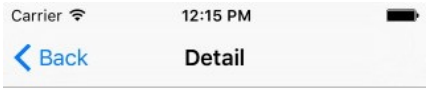

Step 4b - Add basic table code

- 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
}
```

Where are we with Second Animal App?

This screen all works



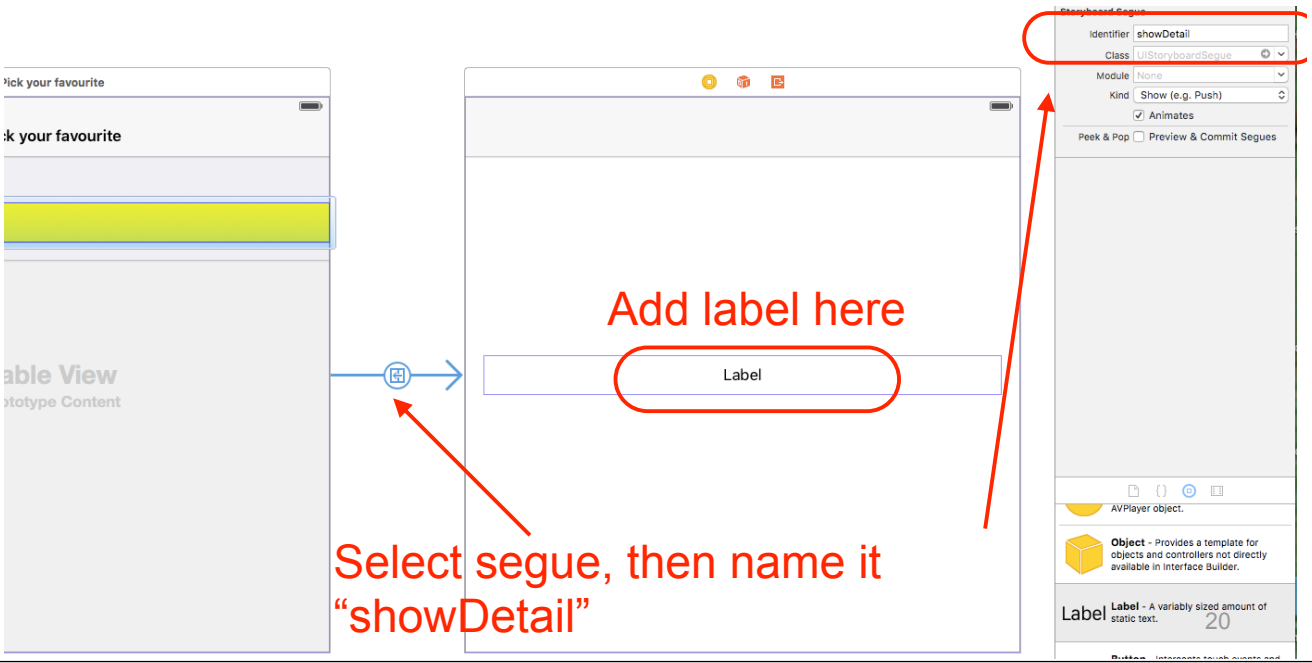
I like Dogs, and my favourite is the Collie

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

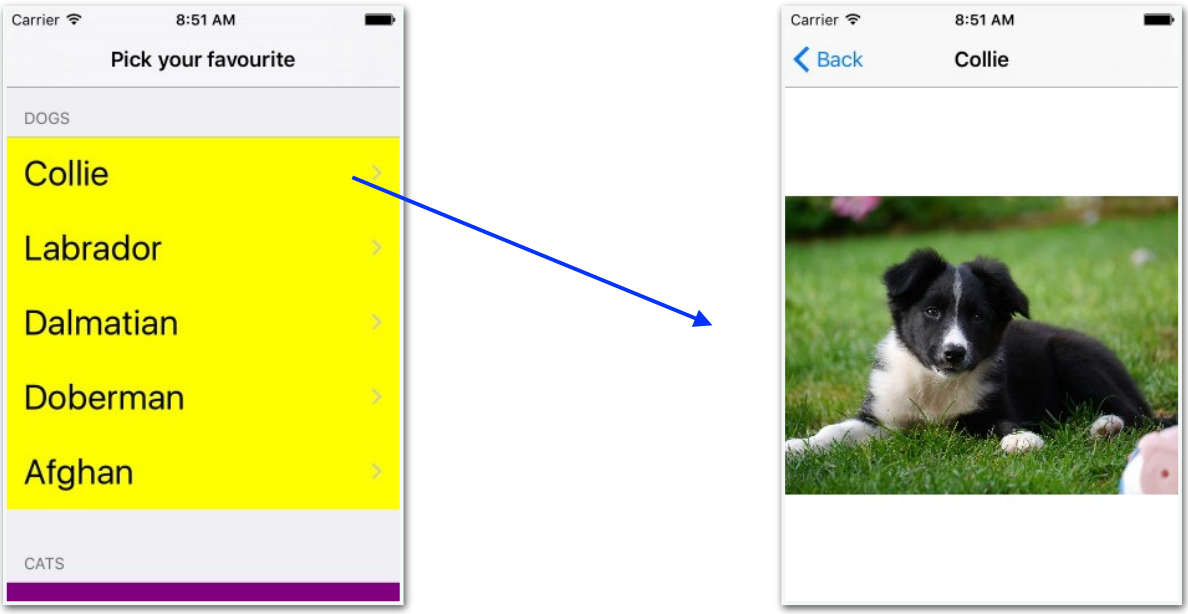
- Could not set the label directly from prepareForSegue as it did not exist at that point. Once the new view controller has been created, can initialise label from passed value

```
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.



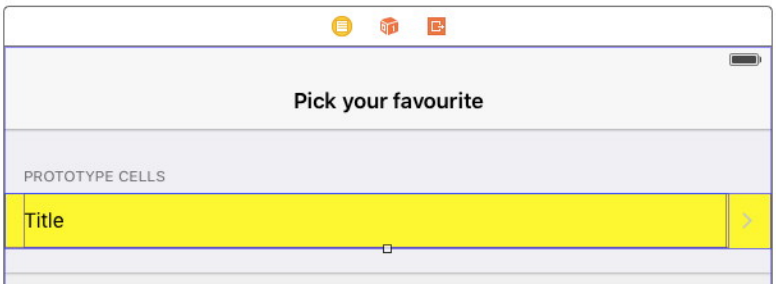
Step 1 - Rearrange AnimalTable

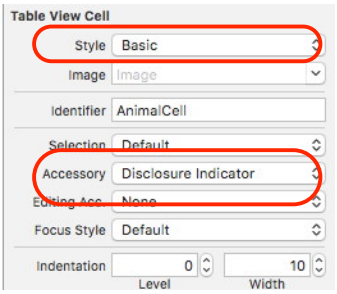
•Change style of AnimalCell to Basic - this will only show a single text. Increase the size of the label text to 28.

•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]
```





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
}
```

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]
```

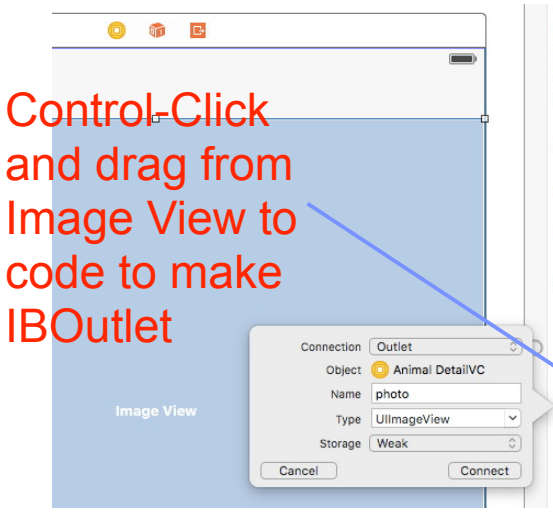
•In prepareForSegue, also change the item that is passed to just be the breed name instead of the long message, and add a title for the screen

```
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

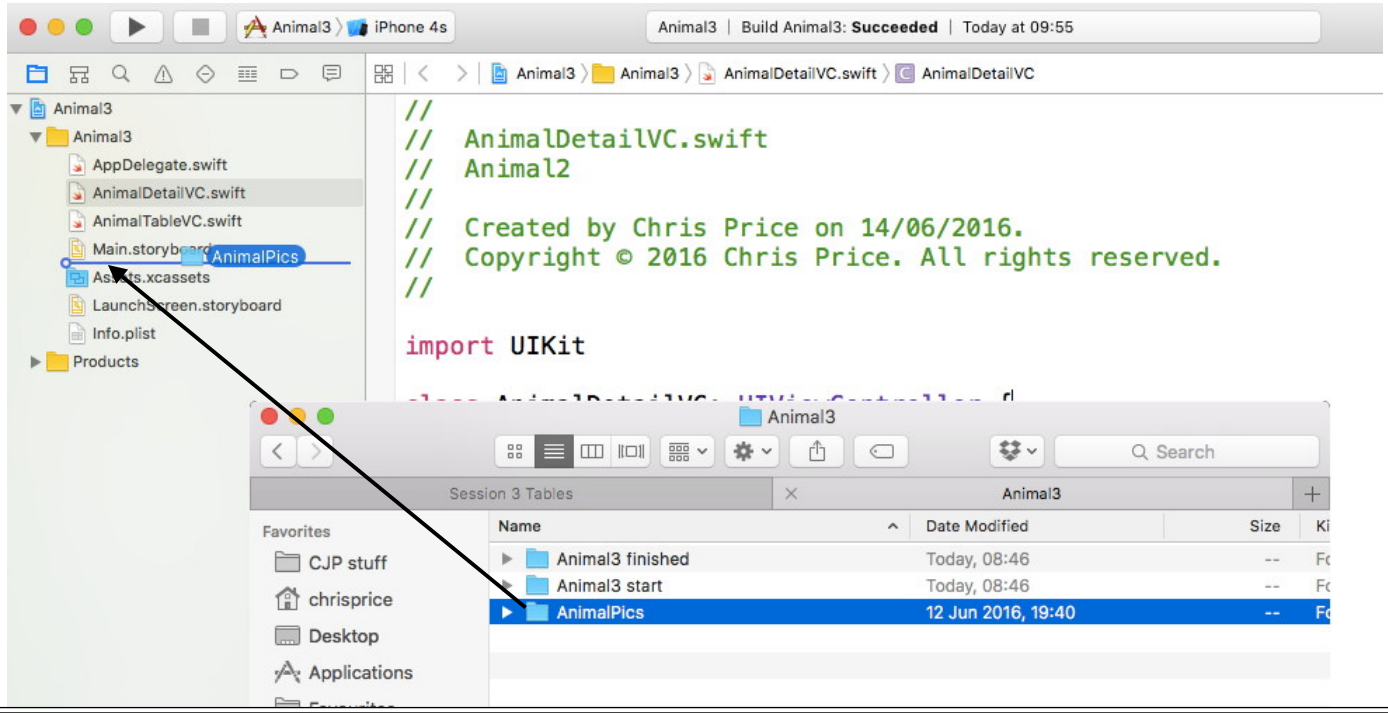
•Delete the label from the detail screen in storyboard
•Add an Image View instead
•Add IBOutlet for the Image View to AnimalDetailVC.swift and call it photo




```
//
//  AnimalDetailVC.swift
//  Animal2
//
//  Created by Chris Price on
    14/06/2016.
//  Copyright © 2016 Chris Price.
    All rights reserved.
//

import UIKit

class AnimalDetailVC:
    UIViewController {

    @IBOutlet weak var favourite:
        UILabel!

    var valueForLabel = ""

    override func viewDidLoad() {
        super.viewDidLoad()
```

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

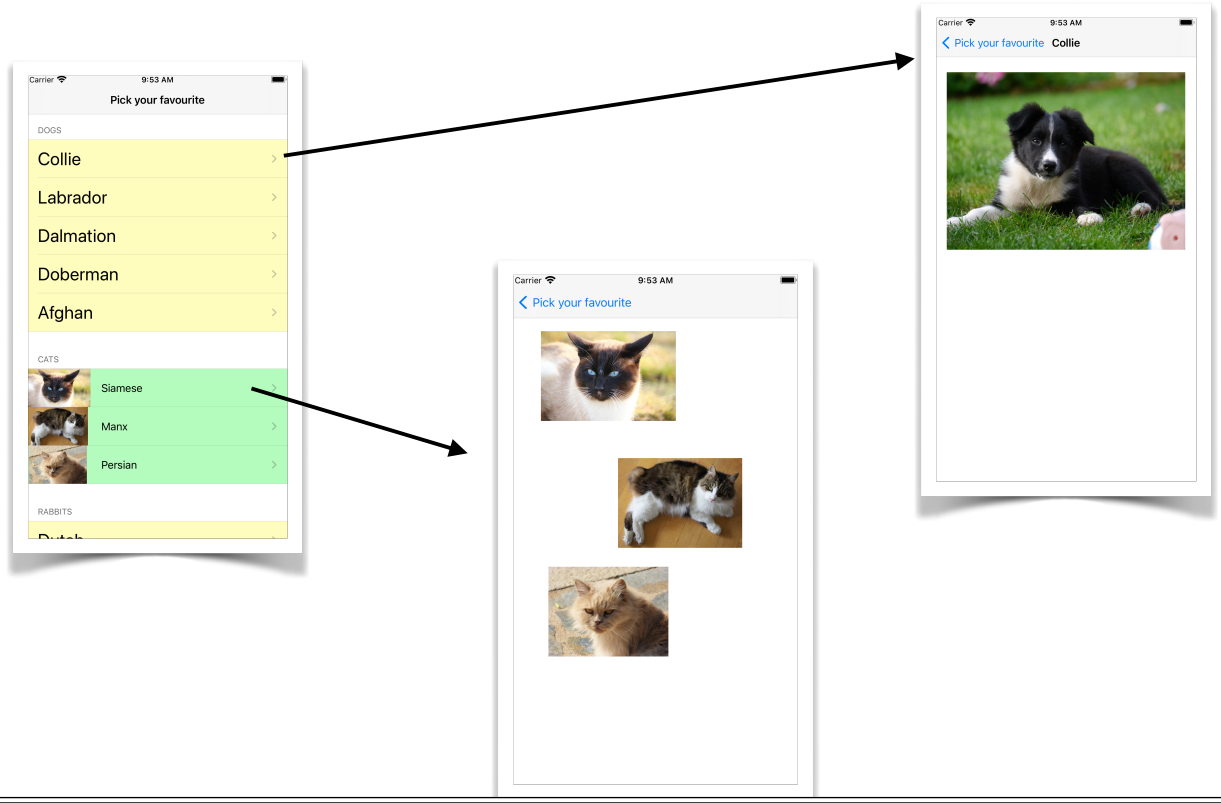
```
@IBOutlet weak var favourite: UILabel!
```

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

Finished!

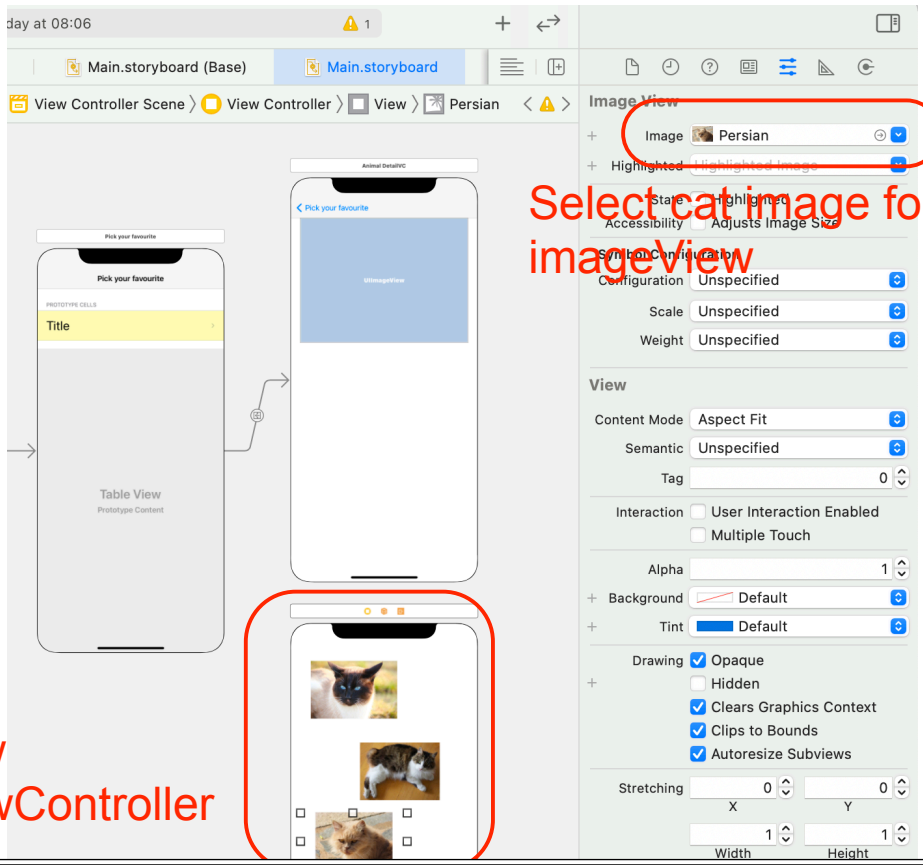
Fourth Animal App
Add custom cells and multiple destinations

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

Step 2 - Make a second cell type

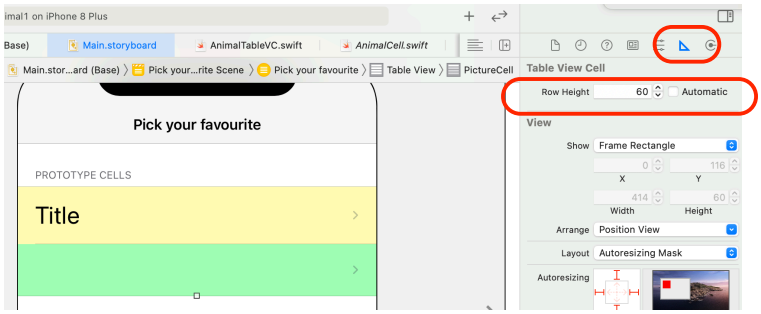
•Select the Tableview in the TableViewController in the storyboard
•Change the number of prototype cells from 1 to 2
•Make the reuse identifier for the second cell “PictureCell”
•Control drag from the new cell to the new ViewController and choose Show
•Colour the new cell green to be able to tell the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

Step 3 - Make cats use new cell

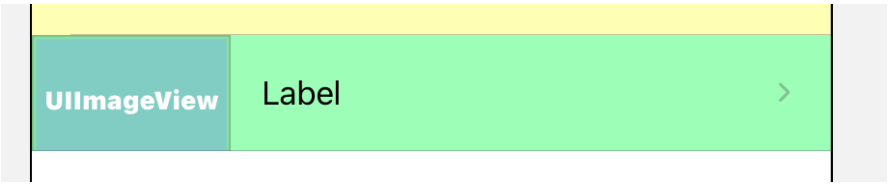
•Select the Tableview in the TableViewController in the storyboard
•Change the number of prototype cells from 1 to 2
•Make the reuse identifier for the second cell “PictureCell”
•Control drag from the new cell to the new ViewController and choose Show
•Colour the new cell green to be able to tell the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

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



Step 5 - Make code for custom PictureCell

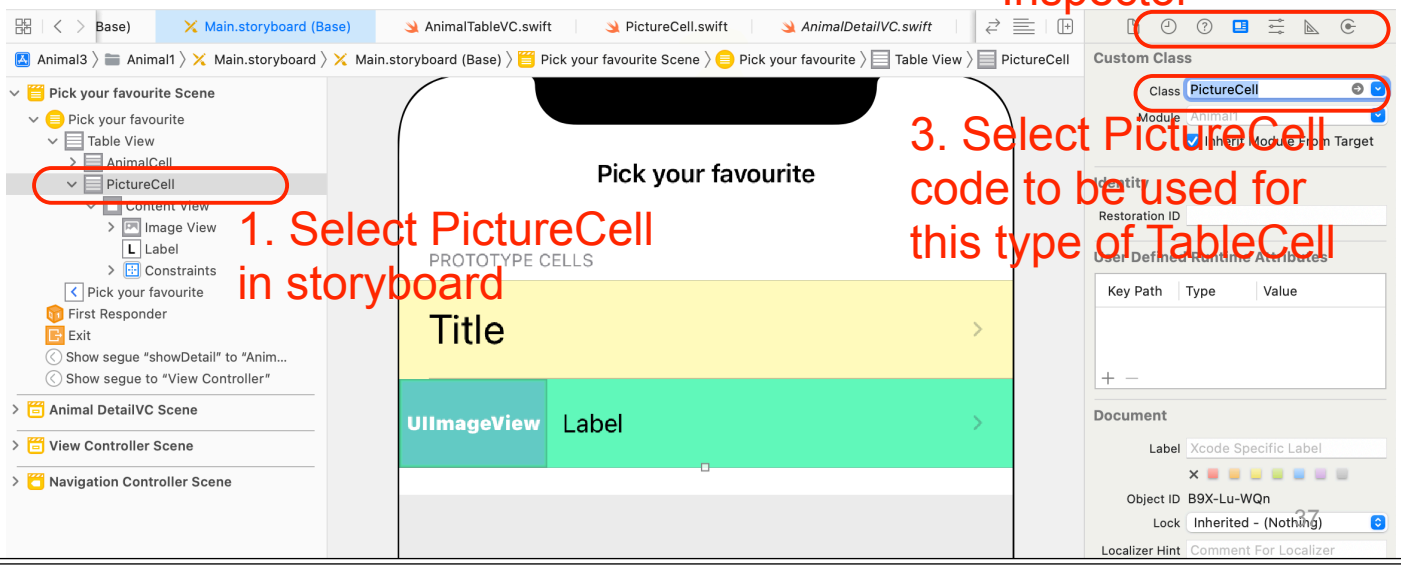
•To have a custom Cell, you need to have some definitions and possibly code to go with it. You do this by creating a UITableCell class associated with the cell, and then making IBOutlets in that class for the custom labels and views

•Create a new Swift file called PictureCell.swift, and declare a UITableView class in it, with outlets for the image and label

```
import UIKit
class PictureCell: UITableViewCell {
    @IBOutlet var photo: UIImageView!
    @IBOutlet var animalName: UILabel!
}
```

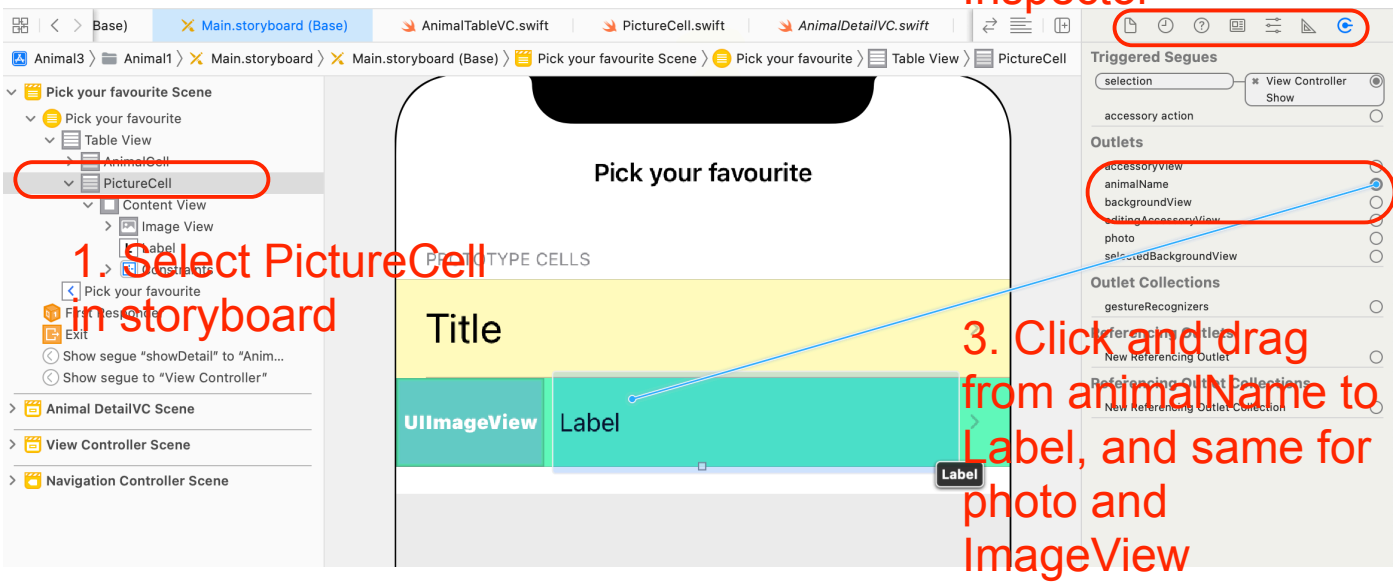
Step 6 - Associate PictureCell code with PictureCell in story board

- We need to link the PictureCell.swift code to the PictureCell view on the storyboard.
- Select the PictureCell in the storyboard, and in the Identity Inspector, link it to PictureCell.swift



Step 7 - Connect the outlets to the cell

- With PictureCell selected in the storyboard, in the Identity Inspector, click and drag from animalName to the label, and from photo to the ImageView.



Step 8 - Add set up code to PictureCell

•Now we have associated the PictureCell code with the Table Cell on the story board, we need some code to set up the details of the cell
•We will add this code to PictureCell.swift, and on the next slide we will call this code from the tableview controller

```
class PictureCell: UITableViewCell {
    @IBOutlet var photo: UIImageView!
    @IBOutlet var animalName: UILabel!

    func configure(photoName: String, animal: String) {
        photo.image = UIImage(named: photoName)
        animalName.text = animal
    }
}
```

New code

Step 9 - 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.configure(photoName: wantedName, animal: wantedName)
            return cell
        } else {
            let cell = tableView.dequeueReusableCell(
                    withIdentifier: "AnimalCell", for: indexPath)
            cell.textLabel!.text =
                        categoryItems[indexPath.section][indexPath.row]
            return cell
        }
}
```

Finished!

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

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

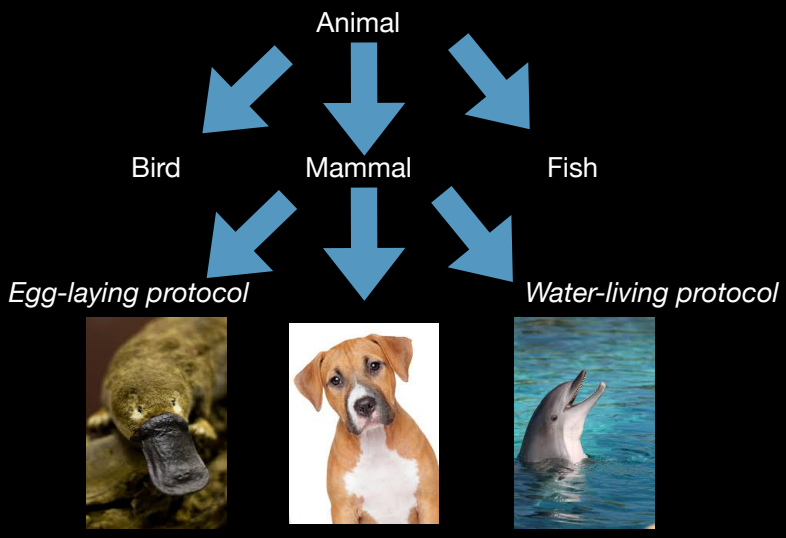
Session 12: Protocols, closures

- 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)

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]
```

```swift
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
  }
}
```

```swift
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

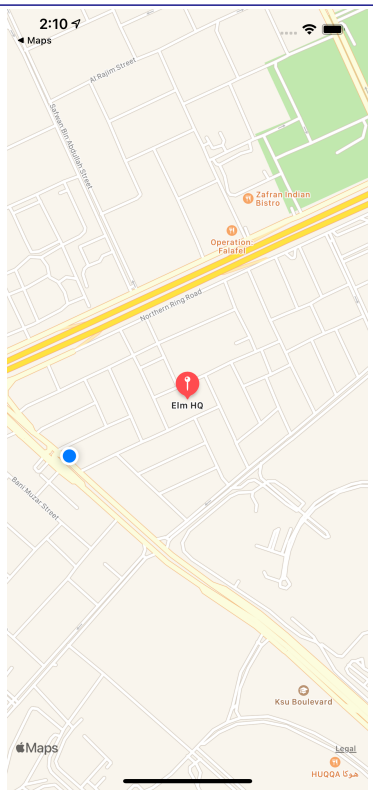
Session 13: Mapping and Location

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

- This material is not covered in the Apple Books

What we are going to build

- A mapping screen which shows landmarks plus our own position



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

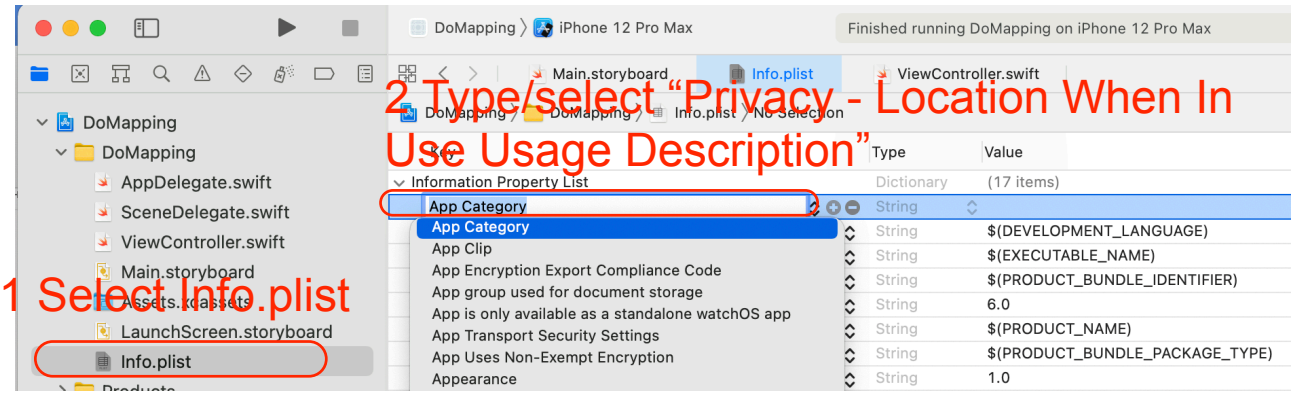
```
class ViewController: UIViewController, CLLocationManagerDelegate {

    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

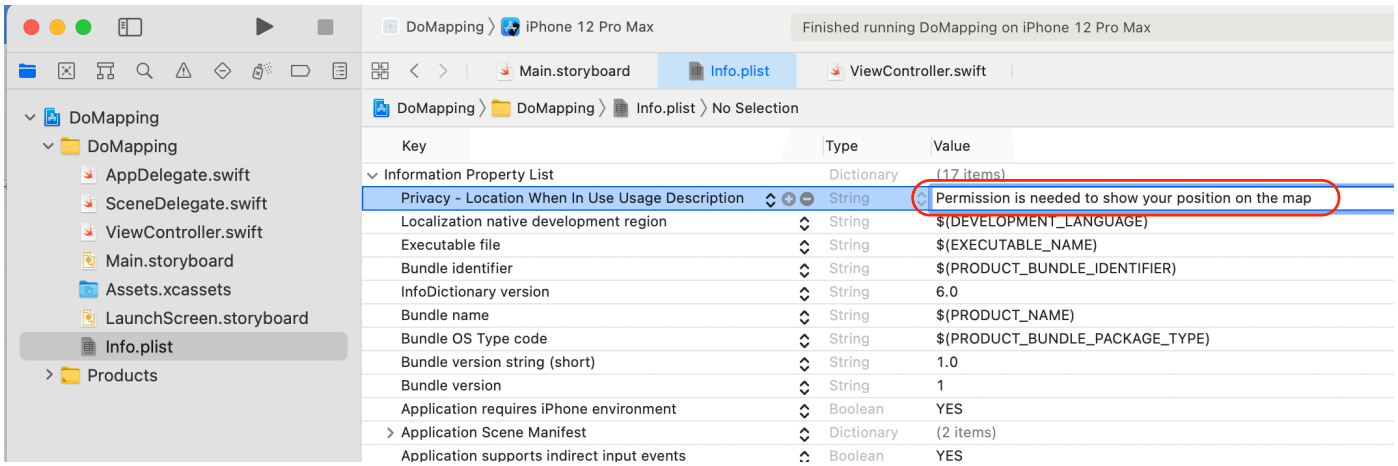
•We now have all the code we need, but when you are doing something you need permission for (using Location, accessing the camera, sending Notifications), you also need to signal that permission in the app's info.plist file



•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).

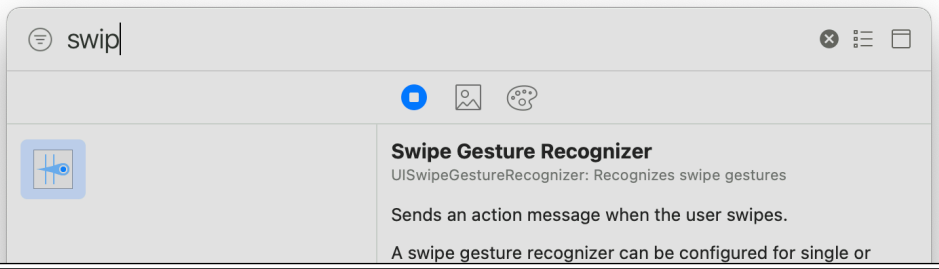
Session 14: Gestures and accelerometer

- 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 301-303 of Develop in Swift Fundamentals

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.



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

Implementing swipes and taps

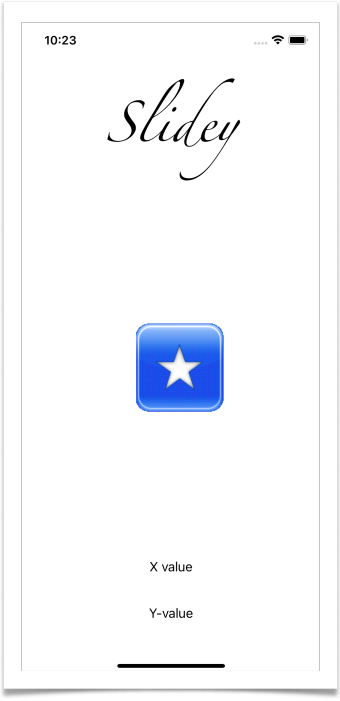
- **Step 3**
- Keep a count of the value of counter, and when there is a right swipe increase it if it is less than 9
- When there is a left swipe, decrease it if it is greater than 0
- Update the counter to reflect value change

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

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



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");
   }
}
```


Turn on only when screen showing

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

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

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

Session 15: Data Storage with Codable

- Application sandbox
- Where to save data in iOS
- Storage choices
- Codable for local state
- *Lab 15: Extended lab bringing together work on tables, data access and storing data locally.*

- This covers lesson 1.7 of Development in Swift Data Collections

Common data patterns in our apps

- We enter data and save it - examples: to-do list, shopping list, blog
- We enter data and save it to the cloud - examples: questionnaires, meter readings
- We use existing local data - examples: our conference app, language learning
- We use cloud-based data - examples: power plant readings, conference app if the data changed over time

Where do we save data in our app?

- Application sandbox - Your App runs in a protected area on the device
- Within this area, there is a defined structure of Directories that you can access
- Where you find what:
- <App Home>/AppName.app - the default Bundle directory
- <App Home>/Documents - backed up to iCloud potentially
- <App Home>/Library/Preferences
- <App Home>/Library/Caches
- <App Home>/tmp
- For simulator you can find <App Home> by printing out its URL

What format do we save data in our app?

- Several different popular techniques for storing and retrieving data on the iPhone:
- JSON - read and write
- Property List Serialization - Mechanism to store a selected number of data types in a property list (plist)
- SQLite - small memory footprint SQL database
- Core Data - more complex system, but it does offer a powerful way to manage data and relationships

What about XML?

- XML is supported in iOS but is not a popular choice of structure - especially since JSON parsing has become so easy
- XML can be read from storage in the same way as we will see for JSON
- You can then call XMLParser with associated parse objects to interpret the XML
- The following tutorial gives an example of doing this:
  - https://www.ioscreator.com/tutorials/parse-xml-ios-tutorial

JSON easiest for simple data

- Codable makes it trivial for flat file records that are made up of data types that are themselves Codable (Int, Double, String, Array, Date etc.)
- More work if you have the kind of relationships in our Conference Data - in those cases you might use Core Data to persist the relationships

Case Study: Creating Speaker Data

- How do we get all the data into our Conference app? At present, we do it by initialising lots of data objects.
- If we encode the data as JSON, we can save it to a file, and then read the file and decode it into data objects in our conference app
- To do the data creation, we could make a ConferenceCreator app (but to simplify things we will just create Speaker data for now)

Step 1 (repeat previous things)

- Make new iOS app called ConferenceCreator
- Delete ViewController and screen
- Add NavigationController from Object library - it comes with a built-in tableviewcontroller
- Make the NavigationController into the initial view controller
- Give the table cell in the tableview controller style Basic, and identifier SpeakerCell

Step 2 - add in provided files

- Add the provided files to the project
  - Speaker.swift is the struct for a Speaker
  - ConferenceData is a singleton for accessing the list of speakers - we will add to it as we go on

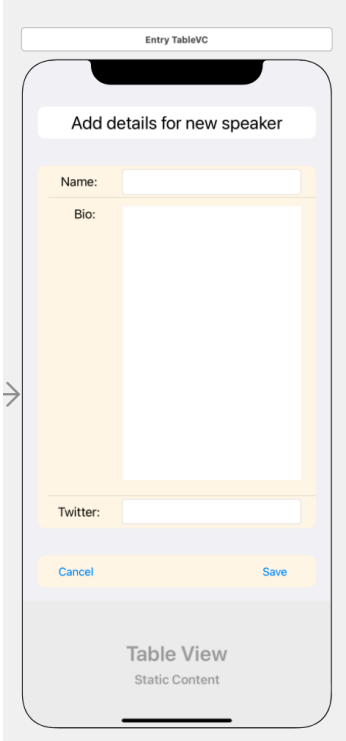
Step 3 - make code for speaker table

- Do File/New/File and create a tableview controller code file called SpeakerTableVC, and associate it with the table view controller
- Fill in the code to populate the table view controller - make a speaker list called speakers and set it:
  - var speakers = ConferenceData.sharedInstance.allSpeakers
- Have one section in the table, with speakers.count cells, and the speaker name in each cell
- Run now, app should show list of (two) speakers

Step 4 - link to another tableview controller

- Add a second table view controller to the main storyboard
- Give the table view static cells, with three sections, and make its style inset grouped
- Set items as shown on next slide

Layout for static tableview controller



First section - one cell, with a label acting as a header

Second section - three cells making up an entry form

First cell has a label “Name:” and a TextField. Set the Textfield options to input a Name, with words capitalised and no spell checking or correction.

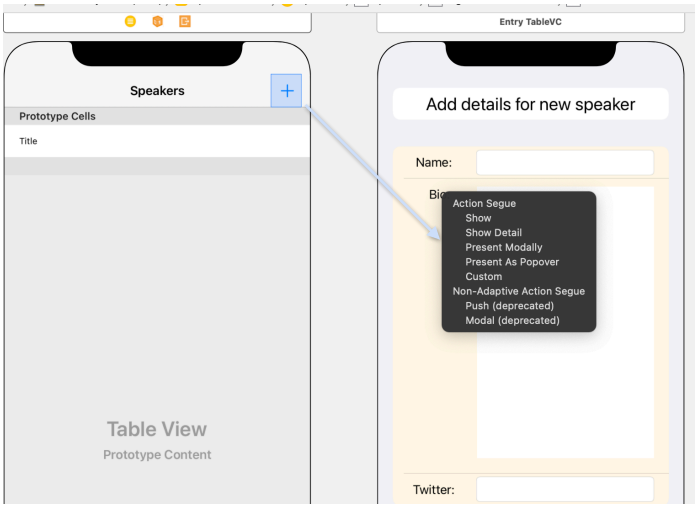
Second cell has a label “Bio” and a TextView for inputting more that one line. Size of cell expanded to allow for lots of text.

Third cell has label Twitter and a TextField

Third section contains two buttons

Step 5 - Link SpeakerTableVC static table

- Add bar button to SpeakerTableVC, and make it a system Add button
- Control-Drag from the Add button to the static table and make the relationship *Presents Modally,* then click on the segue and make the presentation full screen



Step 6 - Code for static table

- We need to make a tableview controller file to associate with the static table - call it SpeakerEntryTableVC
- As the table is static, we do not need to populate it, so can delete all the boiler plate code except viewDidLoad
- We do need to set up outlets for the input fields and actions for the buttons
- We treat them just as we would in a simple viewController

Step 7 - Setting up outputs and save action

- Click and drag from the storyboard to make outputs for speakerName, speakerBio and twitterHandle
- Click and drag from Save button to make action to saveEntry

```
7
8  import UIKit
9
10 class EntryTableVC: UITableViewController {
11
12
13     @IBOutlet weak var speakerName: UITextField!
14
15     @IBOutlet weak var speakerBio: UITextView!
16
17     @IBOutlet weak var twitterHandle: UITextField!
18
19     @IBAction func saveEntry(_ sender: Any) {
20
21     }
22
```


Step 8 - Add code to saveEntry

```
@IBAction func saveEntry(_ sender: Any) {
    // If name is blank, give an alert
    // else add speaker to speaker list and return to main speaker table
    if let name = speakerName.text,
             let bio = speakerBio.text, let twitter = twitterHandle.text {
      let id = name.trimmingCharacters(in: .whitespacesAndNewlines)
      if id == "" {
           let alert = UIAlertController( title: "Sorry",
                           message: "You need to add a name to save a speaker",
                           preferredStyle: .alert)
           alert.addAction(UIAlertAction(title: “OK",
                           style: .default, handler: nil))
           self.present(alert, animated: true, completion: nil)
      } else {
         ConferenceData.sharedInstance.addSpeaker(newSpeaker:
                  Speaker(id: id, name: name, biography: bio, twitter: twitter))
         dismiss(animated: true, completion: nil)
      }
    }
}
```

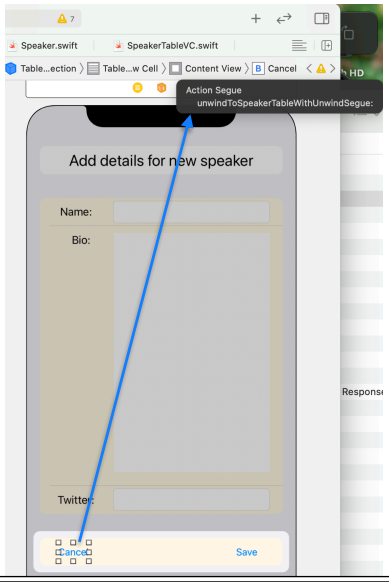
Step 9 - Make the Cancel button return without doing anything

- Add unWindSegue code to SpeakerTableVC
- In the main story board, click drag from the Save button to the top of the screen to link to the unwind segue

Code to put in SpeakerTableVC:

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

Action to link unwind:



Step 10 - Run the app. We can add a speaker, but speaker list does not update - why?

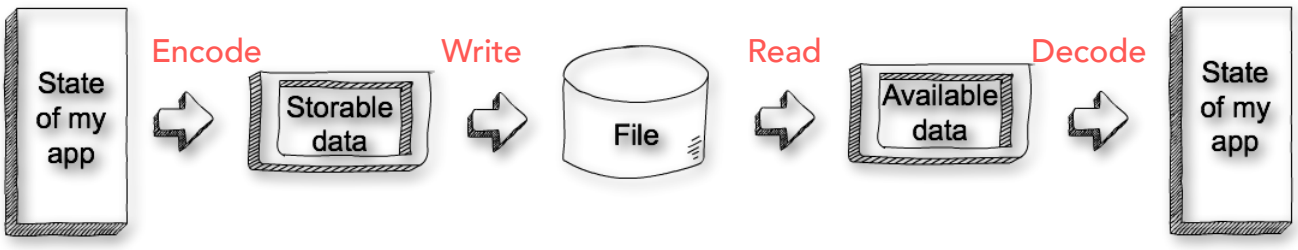
- In code for SpeakerTableVC, the speaker list is set up at the start and not changed when we update the list
- We need to check it and redraw the list whenever we come back to SpeakerTableVC
- Best place to do this is in viewWillAppear:

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    speakers = ConferenceData.sharedInstance.allSpeakers.sorted()
    self.tableView.reloadData()
}
```

Run the app - the list updates, but if we start it again, it goes back to the two original items

- We need to save the list when we update it, and then read it back in when we start up
- Saving is a two stage process - we encode it as JSON, then we write it to the Documents directory

Common workflow



Step 11: Some useful boilerplate file handling code to add to ConferenceData

```
func urlToFileInDocuments( _ fileName: String ) -> URL {
// This function returns the address of the given file in Documents
    let docDirectory = FileManager.default.urls(for: .documentDirectory,
                                                in: .userDomainMask).first!
    let fileURL = docDirectory.appendingPathComponent(fileName)
    return fileURL
}


func fileExistsInDocuments( _ fileName: String ) -> Bool {
// This function tells you whether the named file already exists in Documents
    let fileManager = FileManager.default
    let dirPaths = NSSearchPathForDirectoriesInDomains(
                                    .documentDirectory, .userDomainMask, true)
    let docsDir = dirPaths[0]
    let filepathName = docsDir + "/\(fileName)"
    return fileManager.fileExists(atPath: filepathName)
}
```


Step 12 - Add new method to ConferenceData and call it

```
func saveSpeakers() {
        let readListURL =  urlToFileInDocuments("speakers.json")
        let encoder = JSONEncoder()
        if let data = try? encoder.encode(allSpeakers) {
            //Write the data to backing store.
            try? data.write(to: readListURL, options: .noFileProtection)
            print( "file is at \(readListURL)")
        }
    }
```

- The code above encodes allSpeakers as JSON and then writes it to file speakers.json
- It gives an error as the Speaker struct is not Codable, but as all its attributes are Codable, we just need to add Codable to the header of the Speaker struct
- We can then call saveSpeakers whenever we call method addSpeaker in ConferenceData

Step 13 - Run the app - the console will tell you where speakers.json is saved by the simulator

- If we find that directory (in Finder you choose Go/Go to Folder and provide the folder name), we will see the speakers.json file and can look at its contents
- However, if we restart the app, we will be back to seeing just two speakers, as we do not read back in the list we saved - so we need to do that
- We need to read the list in and decode it when we first open the app (we need to check it exists first, as it won't exist when we open the app the first time)

Step 14 - Give ConferenceData an initialiser that reads the saved file if it exists

- We add the following to ConferenceData
- When the singleton starts, it will look for the speakers.json file on disc, and read it into an array of Speaker objects

```
init(){
    if fileExistsInDocuments("speakers.json") {
        let readListURL =  urlToFileInDocuments("speakers.json")
        if let dataFromFile = try? Data(contentsOf: readListURL) {
            // Decode the plist back to state of program
            let decoder = JSONDecoder()
            if let loadedArray = try? decoder.decode([Speaker].self,
                                        from: dataFromFile) {
                allSpeakers = loadedArray
            }
        }
    }
}
```

Added by request - taking a photo

- See folder 3a
- We could add getting the speaker photos through the app as well as speaker details
- We add a fourth row to the static table, with a button to call the photo
- This invokes a picker to either select an existing photo or take new one
- When saving, we write the file to Documents

Taking photo - provided version

- Provided version uses library images rather than the camera, as simulator does not use camera
- To use real camera, need to change the ImagePicker source to be .camera.
- Also need to get permission to use camera by adding NSCameraUsageDescription to the Info.plist

Comments / expanding this

- We now have an app which creates a list of speakers and saves it
- If we wanted to use this to make all the conference data, we would have arrays of talks and locations and then add all three things into a struct which included the separate arrays
- It would be Codable and can be written /read in one go
- We can add the created JSON file into our previous Conference app with code that reads the data and it should work as it did before

Optional exercise

- We have provided a new version of ConferenceData, and a JSON file called confinfo.json with the same data as before
- Try adding it into the Conference app you made in session 11 during the first three days of the course

Session 16: Unit tests in Xcode

- Identifying what to unit test
- Setting up tests
- Running tests in XCTest
- *Lab 16: Writing Unit Tests for iOS Apps*

- This material is not covered in the Develop in Swift books

Automated Testing in iOS

- Xcode provides templates both for Unit Tests and for UI Testing
- Best strategy is to include them when you start a new project
- They can be added to your project by hand later on, by choosing New/File/Target and then selecting Unit Tests or UI Tests to add to your app
- We will make a new app including testing, and then write tests for a cube function

What do we test in Unit testing

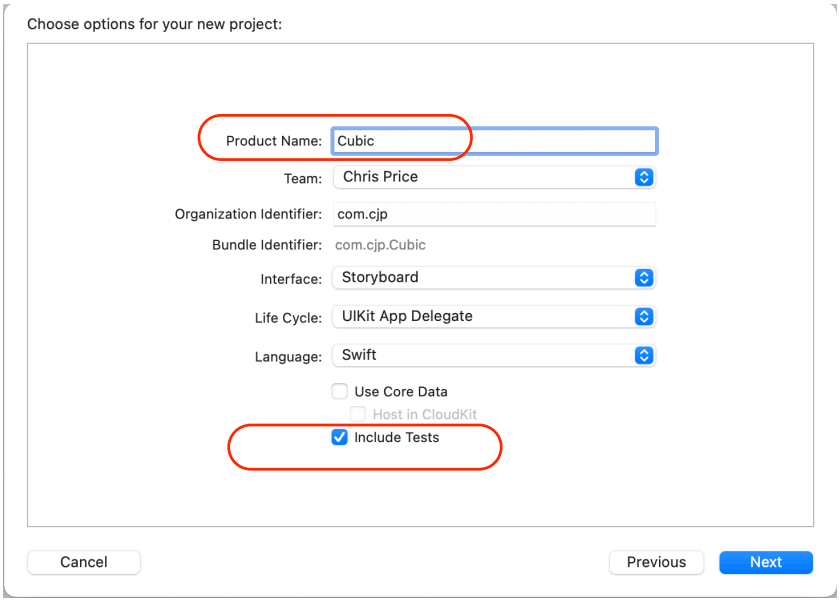
- Unit Tests are appropriate for the back end of your system
  - For example, testing that a cube function returns the correct answer for good values
  - For example, checking that photographs exist for all speakers at the conference

Lets look at easy test situation first

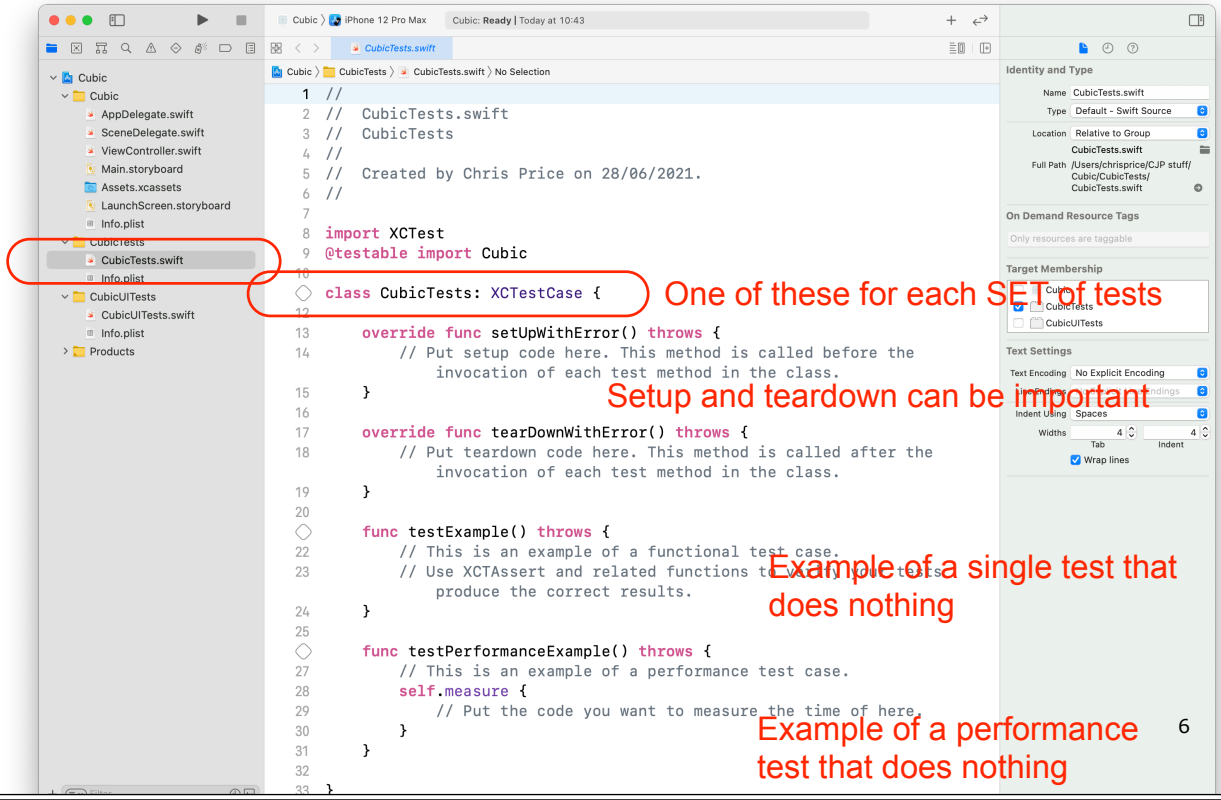
- We will create a program with a math function that returns the cube of a number
- We will write tests to check the function works

Step 1 of testing cube function

- Make new iOS app called Cubic - when choosing options, tick to include testing



Draft template for Unit Testing is provided in CubicTests.swift



Step 2 of testing cube function

- Create Swift file called DoMaths.swift and populate it with a DoMaths class with a stub function for cube as follows:

```
class DoMaths {

    func cube(_ number: Int) -> Int {
        return -1
    }

}
```


Step 3 - write some tests

- In class CubicTests, delete the boilerplate tests and add some real tests:

```
func testCubeOfZero() throws {
    let maths = DoMaths()
    XCTAssertEqual(maths.cube(0), 0)
}

func testCubeOfOne() throws {
    let maths = DoMaths()
    XCTAssertEqual(maths.cube(1), 1)
}

func testCubeOf10() throws {
    let maths = DoMaths()
    XCTAssertEqual(maths.cube(10), 1000)
}
```

Running the tests

- In CubicTests, you will see little diamonds next to the class declaration and each test, to run all tests, either click the class diamond, or type CMD-u

Fixing the failures

- Update cube as shown and rerun the tests and they should pass

```
class DoMaths {

    func cube(_ number: Int) -> Int {
        return number * number * number
    }

}
```

Using setup

- As all our tests need *maths* set up, we can do that in setup and simplify all tests:

```
class CubicTests: XCTestCase {

    var maths: DoMaths!

    override func setUpWithError() throws {
        maths = DoMaths()
    }

    func testCubeOfZero() throws {
        XCTAssertEqual(maths.cube(0), 0)
    }

    func testCubeOfOne() throws {
        XCTAssertEqual(maths.cube(1), 1)
    }

    func testCubeOf10() throws {
        XCTAssertEqual(maths.cube(10), 1000)
    }
```

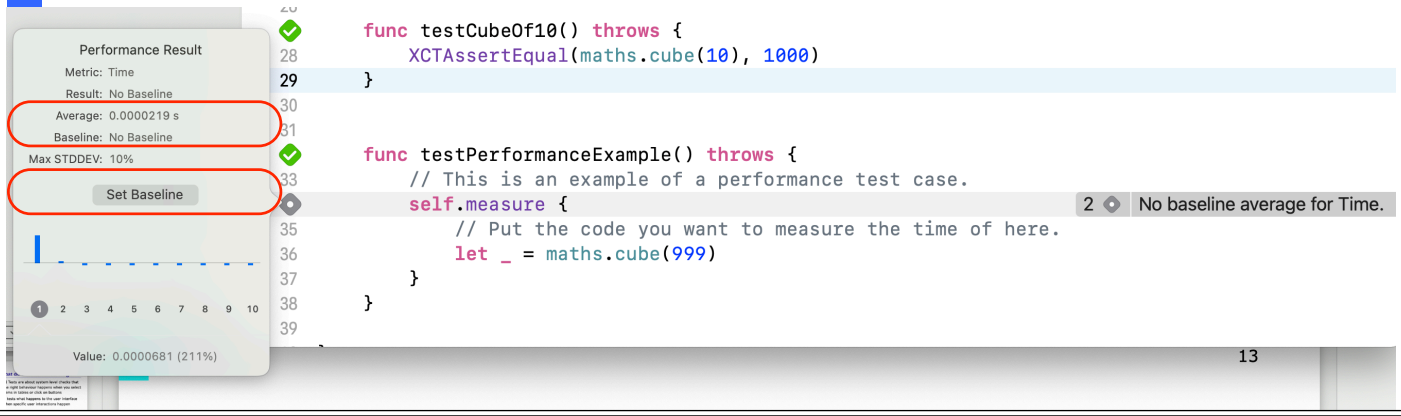

Adding performance tests

- We can take Apple's example performance test and use it to assess our cube function
- When we run this test, we get a grey diamond next to the measurement, and can look at the value

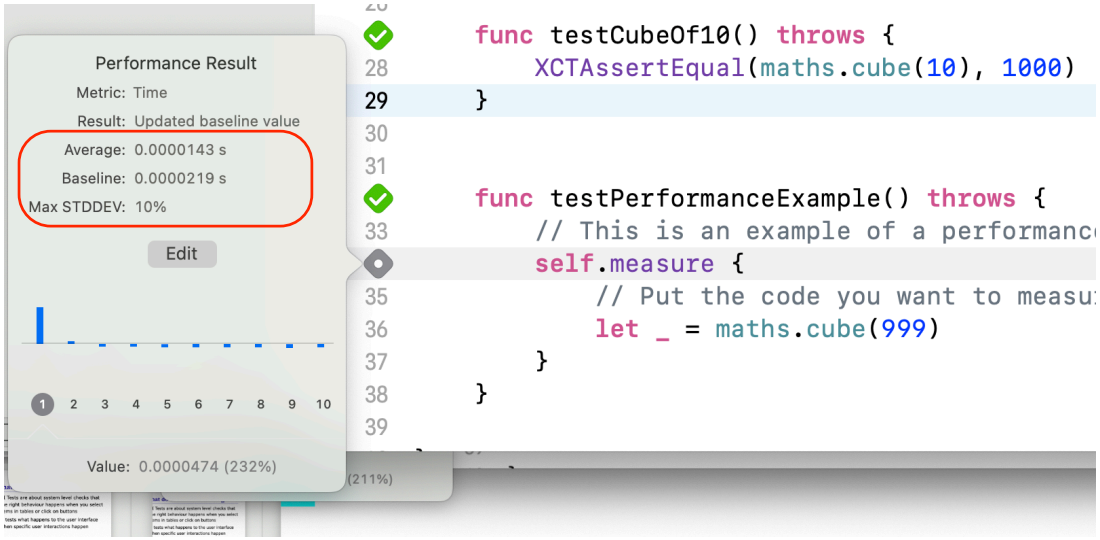
```
func testPerformanceCube() throws {
    self.measure {
        let _ = maths.cube(999)
    }
}
```

Noting performance results

- Clicking on diamond shows timing result
- We can click on Set Baseline to be able to compare future results



Run again to see if changed



Replacing cube

- If we replace cube with the version below, the unit tests will still pass, but performance is shown much worse

```
func cube(_ number: Int) -> Int {
    var answer = 0
    if number == 0 {return number}
    for _ in 1...number {
        answer = answer + number
    }
    return answer * number
}
```

```
31
    func testPerformanceExample() throws {
33      // This is an example of a performance test case.
        self.measure {                                   Time: 0.000 sec (1551% worse)
35          // Put the code you want to measure the time of here.
36          let _ = maths.cube(999)
37      }
38  }
```

Try adding tests to Conference app with JSON

- We want to look at each of the speakers, and check that the photograph exists for that person
- We will need to add unit testing to the conference app that uses JSON and then go through each speaker and check that the speaker photo exists
- Use the conference app with JSON included in this session

What is needed

- Select File/New/Target, and under iOS, select Unit Testing Bundle
- Need to add *@testable import Conference* at the top of the new tests so everything gets imported
- Write a test that iterates through all the speakers and fails if the following test fails for any of them:
  - if let image = UIImage(named: speaker.id)

UI testing in Xcode

- UI Tests are about system level checks that the right behaviour happens when you select items in tables or click on buttons
- It tests what happens to the user interface when specific user interactions happen

What do we test in UI testing

- UI Tests are about system level checks that the right behaviour happens when you select items in tables or click on buttons
- It tests what happens to the user interface when specific user interactions happen

Different from unit tests

- Unit tests can access variables and code within your app
- UI tests can only interface with an app in the way that the user can
- UI tests can only check the state of the user interface after interactions, NOT the state of variables

Theory

- If you click in a UI test, you will see a red record button. Press it and it will start recording your actions
- You can assert what the UI should look like AFTER your actions
- It can then replay your actions
- If the assertion is not true, the test fails

Experience

- It does not seem to work very well
- Sometimes the code generated to reproduce the actions is bad
- Sometimes no code is generated
- Really interesting technology but not ready for serious use

Session 17: Data Storage with Core Data

- Ideas behind Core Data
- Persisting networks of objects
- *Lab 17: Replacing Codable with Core Data*

- This material is not covered in the Develop in Swift books

What is Core Data?

- Core Data is Apple's persistent storage that makes it easy to save and restore a set of objects in memory
- It is especially good if you want to use the same set of objects on different computers - you can link your data model to Cloudkit
- You can also store data in iCloud that all your users can share

How does it work?

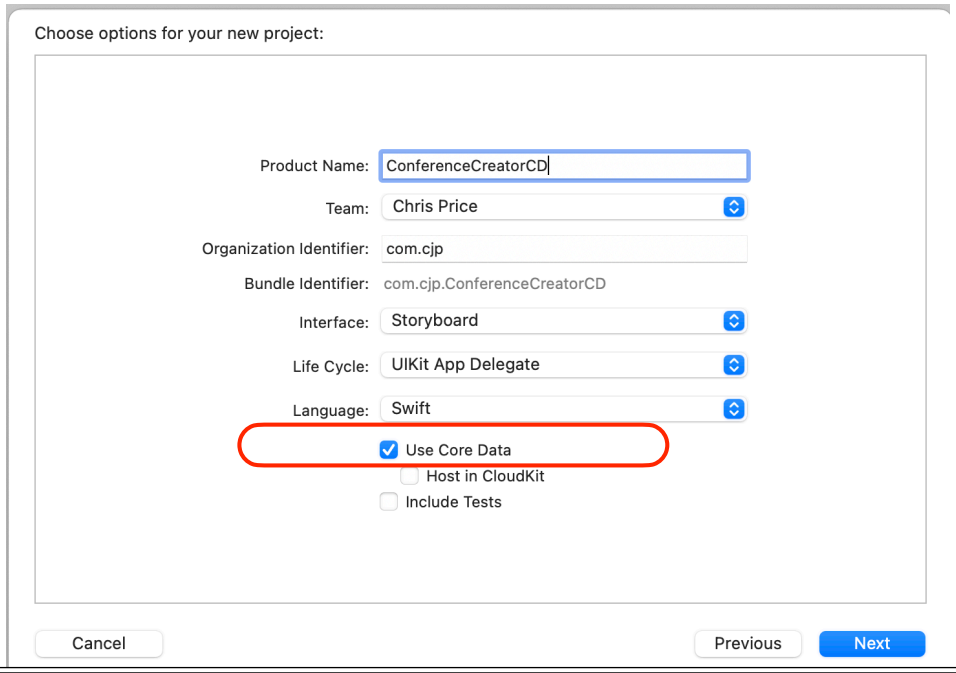
- You specify a set of entities and relationships for your data, and Core Data defines the coding data structures (e.g. the Speaker struct that we have been using in the Conference app)
- You can then create and save such structures - they are saved to a database file (you can choose the underlying format, but default is an SQL database)
- When you start up your app, you can read back in the set of objects that you were dealing with
- We will look at an example with a set of simple objects we have seen before - creating a list of Conference speakers

Conference Creator with CoreData

- We will make a new CoreData project and copy our views and view controllers across - this can be done, but has a few pitfalls
- We will re-implement the Model in CoreData (it was in JSON)

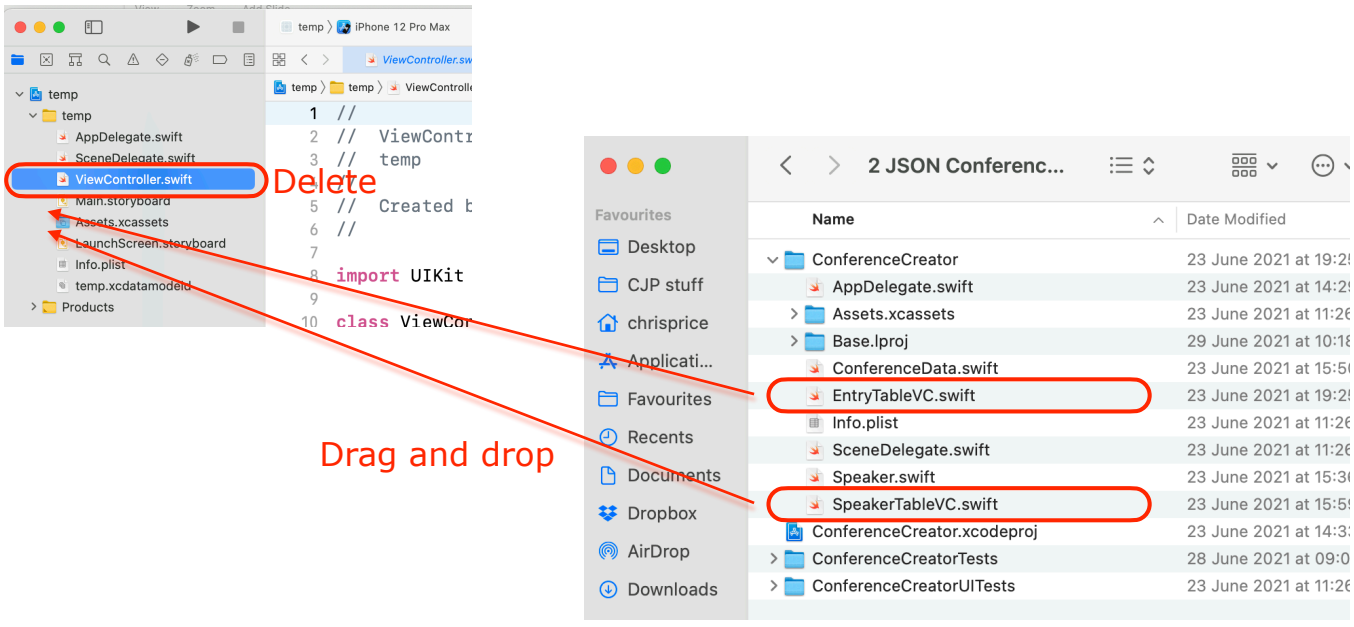
Step 1: Make new app with CoreData

- Make a new app called ConferenceCreatorCD as usual, but tick Use Core Data
- This adds extra code to App Delegate, and a data model



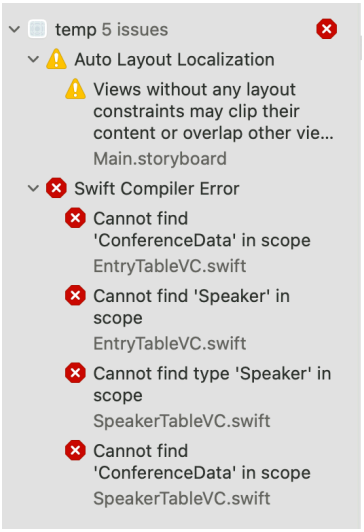
Step 2: Copy code from JSON version

- Delete ViewController and the start screen in Main.storyboard
- Open folder "2. JSON Conference Creator", and drag and drop SpeakerTableVC.swift and EntryTableVC into our new project

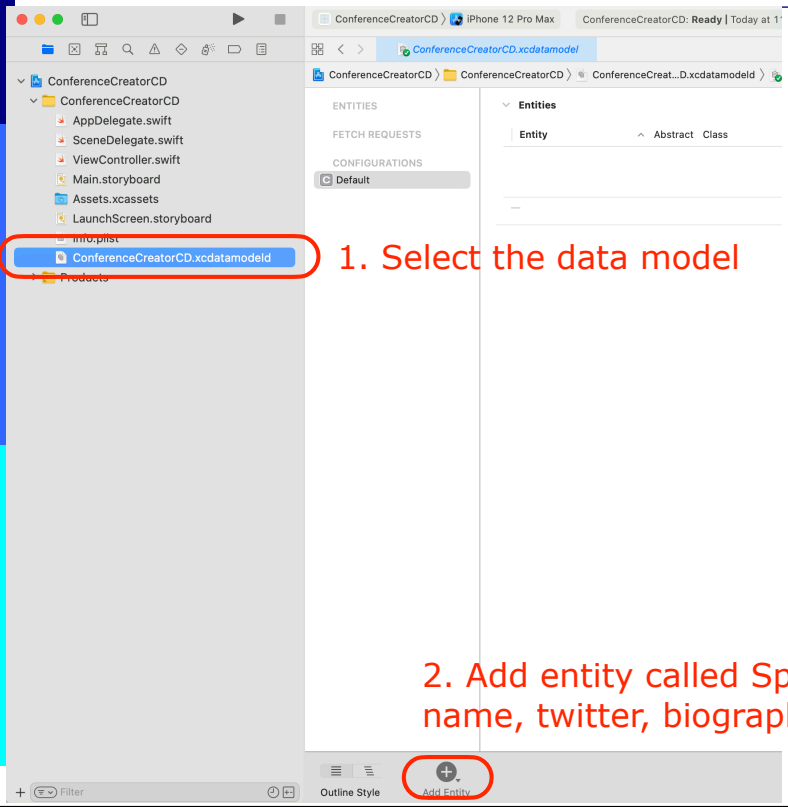


Step 3: Copy screens from old version

- Open JSON version in Xcode, and copy all three of the screens from Main.storyboard
- Paste them into Main.storyboard in the new CoreData project
- Select the NavigationController, and make it the Initial ViewController
- If you try running the app, you get 4 error messages
- We will define Speaker in CoreData, and that will fix two of the errors



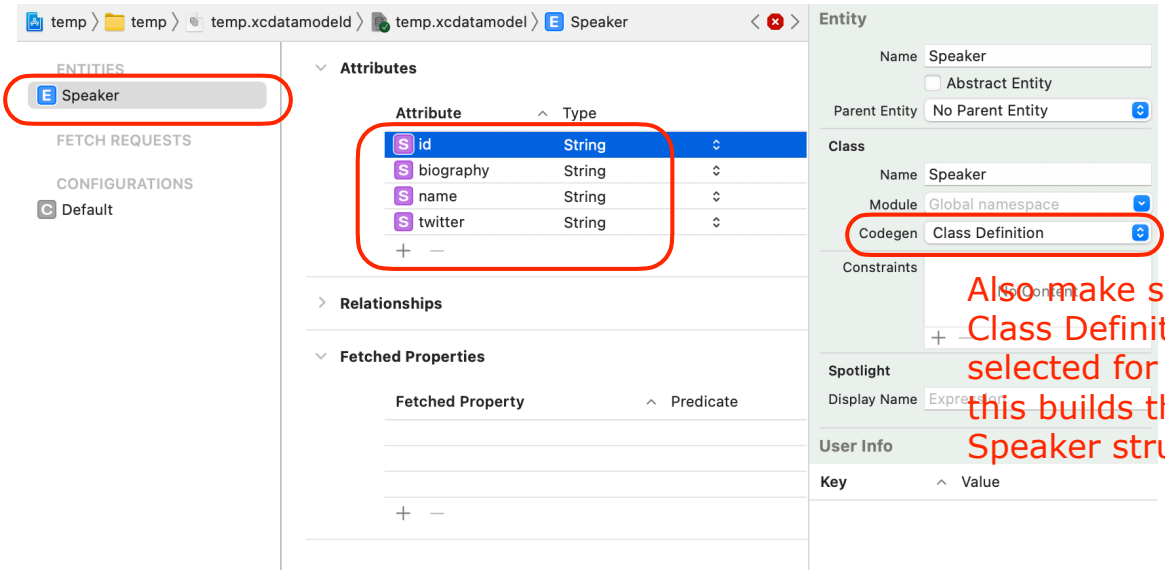
Step 4: Make a Speaker in the data model



1. Select the data model

2. Add entity called Speaker, add attributes - id, name, twitter, biography. Add them all as Strings

After Step 4: model looks like this



Also make sure that Class Definition is selected for Codegen - this builds the Speaker structure

- Try running the app again, should now have 2 error messages

Step 5: Add code to EntryTableVC to save new speaker (also add "import CoreData")

```
func addSpeaker(id: String, name: String, biography: String, twitter: String) {
        // Get the managed context from the AppDelegate
        var managedContext: NSManagedObjectContext!
        let appDelegate = UIApplication.shared.delegate as? AppDelegate
        managedContext = appDelegate?.persistentContainer.viewContext

        // Make the new speaker and add it
        let entity = NSEntityDescription.entity( forEntityName: "Speaker",
                                                 in: managedContext)!
        let speaker = Speaker(entity: entity, insertInto: managedContext)
        speaker.id = id
        speaker.name = name
        speaker.biography = biography
        speaker.twitter = twitter
        try? managedContext.save()

        //Following code is just to find Documents (not really needed)
        let docDirectory = FileManager.default.urls(for: .documentDirectory,
                    in: .userDomainMask).first!
        let fileURL = docDirectory.appendingPathComponent("dummy")
        print( "\(fileURL)" )
}
```

Step 6: Replace line with error in EntryTableVC

- Instead of call to addSpeaker in ConferenceData, add the following code:

```
addSpeaker(id: id, name: name, biography: bio, twitter: twitter)
```

- EntryTableVC should now compile and save a newly created speaker - only one error to fix, in SpeakerTableVC, where we want to read in all saved Speakers

Step 7a: Add code to SpeakerTableVC

- Start by adding "import CoreData" here too
- Next we add the managedContext at the top of the class, as we will reuse it each time we enter the class

```
class SpeakerTableVC: UITableViewController {
  var managedContext: NSManagedObjectContext!
```

New code

- We set up the context once, in viewDidLoad:

```
let appDelegate = UIApplication.shared.delegate as? AppDelegate
managedContext = appDelegate?.persistentContainer.viewContext
```

Step 7b: Add code to SpeakerTableVC

- Replace the call to allSpeakers in viewDidAppear with the following code:

```
let fetch: NSFetchRequest<Speaker> = Speaker.fetchRequest()
fetch.predicate = NSPredicate(format: "id != nil")
do {
     let results = try managedContext.fetch(fetch)
     speakers = []
     for speaker in results {
         speakers.append(speaker)
     }
   } catch let error as NSError {
       print("Could not fetch \(error), \(error.userInfo)")
   }
```


Polishing app

- Our Speaker list used to call sorted, but our auto-defined Speaker is not Comparable
- If we add the following Extension to Speaker, then we can sort the list

```
extension Speaker: Comparable {
    public static func < (lhs: Speaker, rhs: Speaker) -> Bool {
        if let leftname = lhs.name, let rightname = rhs.name {
            return leftname < rightname
        }
        return false
    }
}
```

Things to note

- We have created a model, and CoreData takes care of the storing of objects
- We have seen reading and writing of objects
- We have seen how to extend an existing class (chapter 2.2 of Data Collections has more on this)
- Our example was simple - we can add relationships as well as attributes, and have linked objects
- CoreData is a big enough subject to have whole books on it - they are referenced in the course reading list

Session 18: Practical Animations

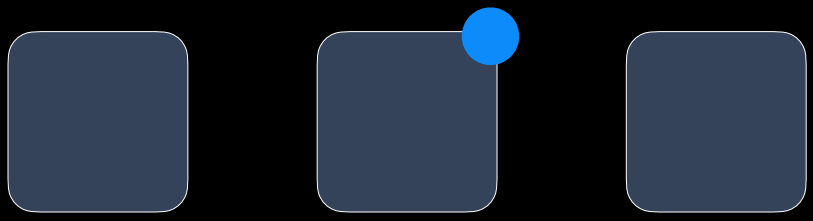
- How animations are used in iOS
- Delighting users with animations
- Types of animation that are possible
- *Lab 18: Building animations*

- This covers lesson 2.3 of Development in Swift Data Collections

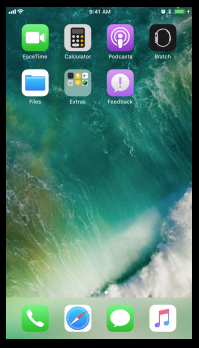
Animations in iOS

- As with Powerpoint, a lot is too much!
- Built-in animations already do some of this
  - When app starts / ends
  - When "Show"ing a new screen
  - When leaving a screen
  - When showing a modular screen
  - When scrolling
  - When reaching the end of a table

Why animate?
Direct the user's attention

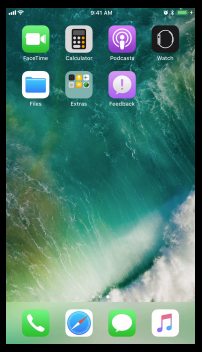


Why animate?
Keep the user oriented



Why animate?

Connect user behaviours



What can be animated?

`UIView`

- `alpha`
- `backgroundColor`
- `bounds`
- `center`
- `frame`
- `transform`

UIView animation methods

```
animate(withDuration:animations:)

animate(withDuration:animations:completion:)

animate(withDuration:delay:options:animations:completion
:)

animate(withDuration:delay:usingSpringWithDamping:initia
lSpringVelocity:options:animations:completion:)
```

Animation closures

animate(withDuration:animations:completion:)

```
UIView.animate(withDuration: 2.0, animations: {
    //animation closure
    viewA.alpha = 0.0
}) { (_: Bool) in
    //completion closure
    UIView.animate(withDuration: 2.0, animations: {
        //second animation closure
        viewB.alpha = 1.0
    })
}
```

Where might you add animations

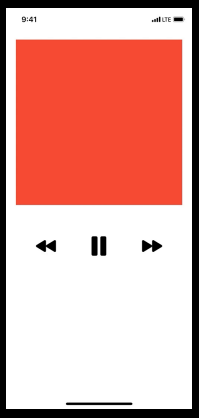
- To show transitions
  - e.g. finishing a test
- To indicate completion of a transaction
  - e.g. adding an item to a basket
- To get the user's attention
  - e.g. because some mail has arrived

Practical Animation

Learn how to use the UIView class and closures to add animations that improve the presentation and functionality of your apps.

Create a wireframe—just the views, without actual functionality—of the Now Playing screen in the Music app.

(Page 349 of Data Collections book)



Session 19: Web browsing, web services

- Integrating web browsing into your app
- Calling web services and receiving data asynchronously
- Concurrency and Grand Central Dispatch
- Processing data and updating screens
- *Lab 19: Reading and processing data from a web service*
- This covers lessons 2.4, 2.5, 2.6 of Development in Swift Data Collections

Easiest web access - web browser within your app

- If you want to show a web page, you can link to Safari to load it
- Loading it in a web browsing screen within your app is as easy and a better user experience

Easiest web access - web browser within your app

- If you want to show a web page, you can link to Safari to load it
- Loading it in a web browsing screen within your app is as easy and a better user experience
- Lets make an app that accesses Elm's web page - first make a simple app, and put a button on the home page

Add code to ViewController

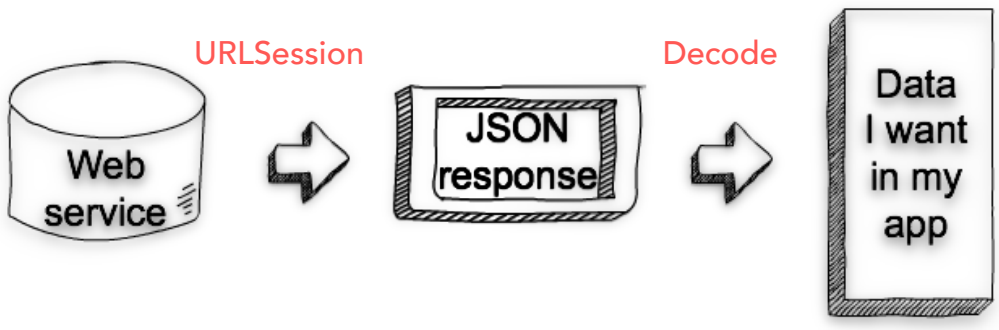
- Add following code to button action - also need to *import SafariServices*

```
@IBAction func loadURL(_ sender: UIButton) {
    // Do any additional setup after loading the view.
    if let pageURL = URL(string: "https://www.elm.sa/") {
       let browser = try SFSafariViewController.init(url: pageURL)
       present(browser, animated: true, completion: nil)
    }
}
```

Optional extra exercise

- Add a segmented control to allow the user to switch between the Arabic version of the web site and the English version of the site.

Next level - load data from web service



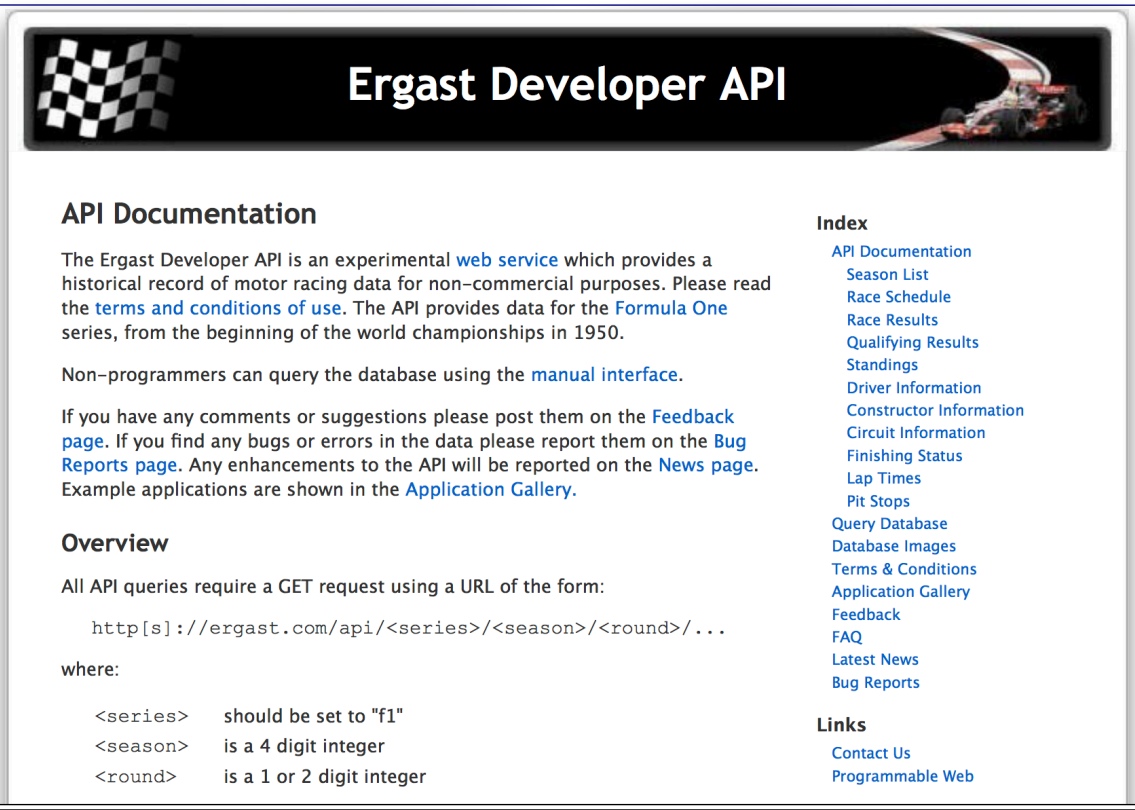
What do we have to do (Step 1)

- Make a URLSession on the URL that addresses the JSON web service that you want to access, and call "resume" to run it. The data will come back asynchronously, and the closure on the URLSession declaration tells you how to deal with it

What do we have to do (Step 2)

- Identify structure of the JSON and make a matching codable struct that parallels it, providing properties only for the JSON fields that you want to keep.
  - You decode the data into the codable structure you have defined
  - The decoder sorts through the JSON and pulls out the data you want into your codable struct

Our example web service



We will use a playground to explore getting the data and decoding it

- Playgrounds execute once synchronously by default, but we can make it handle asynchronous code by adding this at the top:

```
import PlaygroundSupport

PlaygroundPage.current.needsIndefiniteExecution = true
```

Step 1: Get the data with URLSession

- We can get info on all 2016 Grand Prix races with a web service call:

```
if let url = URL(string: "http://ergast.com/api/f1/2016.json") {
    let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
        if let goodData = data {
            print( String(decoding: goodData, as: UTF8.self))
        }
    }
    task.resume()
}
```


HERE'S ONE I PREPARED EARLIER...


```
{"MRData":
    {"xmlns":"http:\/\/ergast.com\/mrd\/1.4",
     "series":"f1",
     "url":"http://ergast.com/api/f1/2016.json",
     "limit":"30",
     "offset":"0",
     "total":"21",
     "RaceTable":
        {"season":"2016",
         "Races":[
            {"season":"2016",
             "round":"1",
             "url":"https:\/\/en.wikipedia.org\/wiki\/2016_Australian_Grand_Prix",
             "raceName":"Australian Grand Prix",
             "Circuit":
                {"circuitId":"albert_park",
                 "url":"http:\/\/en.wikipedia.org\/wiki\/Melbourne_Grand_Prix_Circuit",
                 "circuitName":"Albert Park Grand Prix Circuit",
                 "Location"
                    {"lat":"-37.8497",
                     "long":"144.968",
                     "locality":"Melbourne",
                     "country":"Australia"}
                },
            "date":"2016-03-20",
            "time":"05:00:00Z"
            },
            {"season":"2016",
             "round":"2", ….

            }, ….
              ]
        }
    }
}
```


Part of this big MRData record

Data I actually want

In a record with a bunch of other stuff

There's an array of them in a field called Races

Part of a record called RaceTable

PICK OUT THE DIFFERENT LEVELS

4. Record attribute called MRData contains RaceTable

3. Record attribute called RaceTable contains Races

2. Array attribute of Race called Races

1. Race


```
{"MRData":
    {"xmlns":"http:\/\/ergast.com\/mrd\/1.4",
     "series":"f1",
     "url":"http://ergast.com/api/f1/2016.json",
     "limit":"30",
     "offset":"0",
     "total":"21",
     "RaceTable":
        {"season":"2016",
         "Races":[
            {"season":"2016",
             "round":"1",
             "url":"https:\/\/en.wikipedia.org\/wiki\/2016_Australian_Grand_Prix",
             "raceName":"Australian Grand Prix",
             "Circuit":
                {"circuitId":"albert_park",
                 "url":"http:\/\/en.wikipedia.org\/wiki\/Melbourne_Grand_Prix_Circuit",
                 "circuitName":"Albert Park Grand Prix Circuit",
                 "Location"
                    {"lat":"-37.8497",
                     "long":"144.968",
                     "locality":"Melbourne",
                     "country":"Australia"}
                },
            "date":"2016-03-20",
            "time":"05:00:00Z"
            },

            {"season":"2016",
             "round":"2", ….

            }, ….
              ]
        }
    }
}
```


Structure to get wanted items

- Declare the structure I actually want - it is in a record with lots of other fields, but the JSON decoder will just find the ones we want:

```
struct Race: Decodable {
    let round: String
    let url: URL?
    let raceName: String
}
```

Build outwards to match whole data file

- There is an array of Race records called Races within a RaceTable, within an MRData

```
struct Schedule: Decodable {
    struct MRDataStruct: Decodable {
        struct RaceTableStruct: Decodable {
            let Races: [Race]
        }
        let RaceTable: RaceTableStruct
    }
    let MRData: MRDataStruct
}
```


Step 2: Decode the data

- Having added the definition for a Race and a Schedule to our playground, we can replace the print statement after ***if let*** *goodData = data* with code to decode the JSON into a Schedule

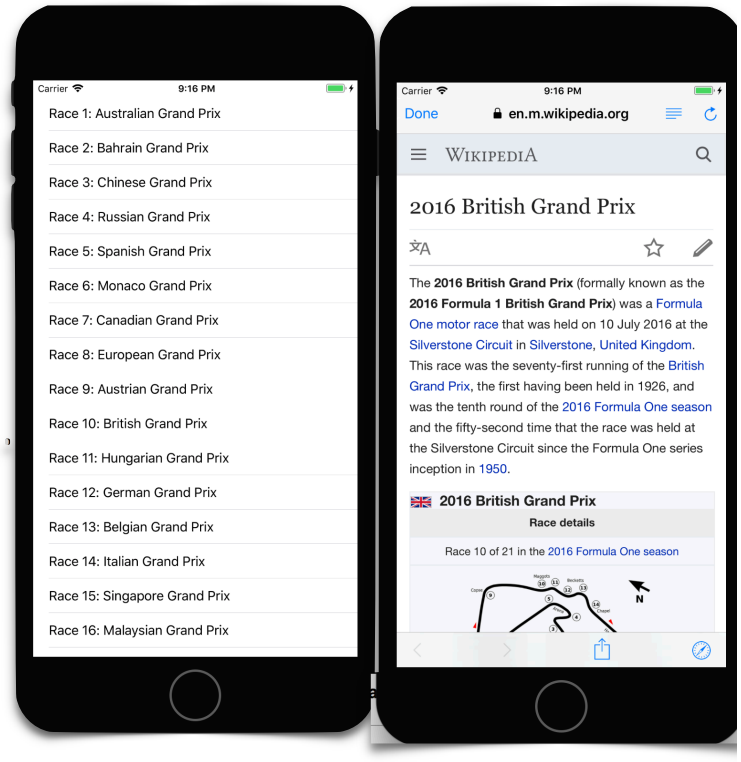
```
let decoder = JSONDecoder()
let schedule = try! decoder.decode(Schedule.self, from: goodData)
let races = schedule.MRData.RaceTable.Races
for race in races {
      print( "Round \(race.round): \(race.raceName)")
}
```

Several refinements are possible

- Error handling can be improved
- Explicit decoders can be written where you specify in detail how to interpret data
- Naming mappings can also be declared if you do not like the names of fields in the JSON
- If an item of data may be omitted in the JSON, then make the field Optional, and test whether a value was read in when you wish to access it

Lets make an app from this data

- We can add this code in viewDidLoad in a tableviewcontroller, and make an app which is a list of the 2016 Grand Prix circuits, and when we select one of them, it loads a web page from the URL associated with that circuit



What you need to do (main screen)

- New app, make initial screen a tableviewcontroller
- Declare an array of Races at the top level, and read data into the array in viewDidLoad (using code from our last playground)
- Use the array of Races in the usual way when populating tables

Problem!

- When we run the app, no data appears on screen
- Depending on how you have written it, the app may crash as well
- We can see better what the problem is if we initialise our array of Races as follows:
  - Var races = [Race(round: "", url: **nil**, raceName: "Waiting for data")]

The table is populated BEFORE we get the JSON data

- viewDidLoad runs before we populate the array, but the data is not obtained at that point
- viewDidLoad runs on the main thread, but the decoding is done asynchronously on a separate thread, AFTER viewDidLoad and numberOfSections and the other calls terminate

How it goes

Main thread

Other thread

Time

Initialise races

viewDidLoad

numberOfSections

numberOfRowsInSection

cellForRowAt Indexpath

(Stop)

Data received - do the code in the closure in URLSession

Need to update table now

We know how to update a table

- We call self.tableView.reloadData()
- However, this will fail (try it)
- All UI updating MUST be done on the main thread
- We can do this in the closure by explicitly telling the system to run reloadData on the main thread:

```
DispatchQueue.main.async {
   self.tableView.reloadData()
}
```


Finally we need to show a web browser with the URL if the user selects a table cell

- We can do this by overriding the tableView function didSelectRowAtIndexPath
- This lets us take an action when a specific cell is chosen

```
override func tableView(_ tableView: UITableView,
        didSelectRowAt indexPath: IndexPath) {
    if let url = races[indexPath.row].url {
        let myWebVC = SFSafariViewController(url: url)
        present(myWebVC, animated: true, completion: nil)
    }
}
```

We have successfully built an app that deals with asynchronous inputs

Next we will look at the more general issues of concurrency

Concurrency in Swift: why and how

While building our web app, we called:

```
DispatchQueue.main.async {
    self.tableView.reloadData()
}
```

- This was necessary because we were working on a background thread
- How did we get on a background thread in the first place?

DispatchQueue is part of more general concurrency provided by Grand Central Dispatch (GCD)

- URLSession is built on top of GCD for the special case of getting web data, and hides much of GCD
- Essentially URLSession calls GCD to set up code that works on a background thread to handle the case of receiving data from a web request
- We can see the need for this if we use a more explicit call to get the data that does not use GCD

Calling

- The example in folder 6 of Session 19 calls Data(contentsOf:) instead of URLSession - this call waits for a reply, so the app freezes until that happens - using GCD gets us around this

```
if let url = URL(string: "https://ergast.com/api/f1/2016.json") {
        if let data = try? Data(contentsOf: url) {
             let decoder = JSONDecoder()
             let schedule = try! decoder.decode(Schedule.self, from: data)
             self.races = schedule.MRData.RaceTable.Races
             self.tableView.reloadData()
}
```

What is Grand Central Dispatch

- GCD is Apple's scheme for prioritising tasks
- It allows you to spin off a task, and say how important it is
- It also allows iOS to make as much use as possible of multiple cores, as tasks can then be assigned to different processors

Levels of importance in GCD

- GCD allows you to specify the Quality of Service (QoS) you expect for a task
- There are five levels of service
  - User Interactive - this is the top priority, as you want the interface to be responsive
  - User Initiated - next level, for tasks that the user is waiting for, e.g. processing a photo where the user wants to see the results
  - Utility - for tasks the user is not actively waiting for, such as checking a remote database for updates
  - Background - for tidying up tasks
  - Default - where activities such as URLSession would run if nothing else was specified. It is between User Initiated and Utility
- You can think of the levels of service as being priority queues - user initiated tasks will run before default tasks, before Utility tasks, etc.

Example use in Conference app

- We have included all of the photos of speakers within our conference app
- What if we did not do that, but instead had to download the photos from the Internet?
- We would not want to do this on the main thread
- We could call Data(contentsOf:) on the Utility thread, once for each photo that is not presently stored
- Folder 7 gives an example of this
- In practice, we would probably use URLSession to do this, but what is happening behind the scenes is similar

Example use in Conference app

```
func loadSpeakerPhotos() {
    let accessStem = "https://github.com/chrisinaber/iosdevforelm/raw/main/Speakers/"
    for speaker in allSpeakers {
        let fileName = speaker.id + ".jpg"
        if !fileExistsInDocuments(fileName) {
            let fileString = accessStem + fileName
            if let url = URL(string: fileString ) {
             DispatchQueue.global(qos: .utility).async {
                if let data = try? Data(contentsOf: url) {
                    let photoURL =  self.urlToFileInDocuments(fileName)

                    //Write the data to backing store.
                    try? data.write(to: photoURL, options: .noFileProtection)
                    print( "file is at \(photoURL)")
                  }
                }
            }
        }
    }
}
```

This is about to get easier

- Many calls to dispatch queue end up embedded in each other
- Example: if we were reading the json with speaker names in it, then we would embed the call to get the photos inside the successful call to get the json
- This will not be necessary with iOS 15, because there is new syntax for handling this kind of thing
- It will only work with the new operating system though

Session 12: Protocols, 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)

Unit 2—Lesson 2: Extensions

Extensions

```
extension SomeType {
  // new functionality to add to SomeType goes here
}
```

Adding computed properties

```
extension UIColor {
  static var random: UIColor {
    let red = CGFloat.random(in: 0...1)
    let green = CGFloat.random(in: 0...1)
    let blue = CGFloat.random(in: 0...1)

    return UIColor(red: red, green: green, blue: blue, alpha: 1.0)
  }
}
```

Adding instance or type methods

Apple ➔ Apples

Song ➔ Songs

Person ➔ People

Tennis court ➔ Tennis courts

Adding instance or type methods

```
extension String {
  mutating func pluralize() {
    // Complex code that modifies the current value (self) to be plural
  }
}

var apple = "Apple"
var person = "Person"
apple.pluralize()
person.pluralize()

print(apple)
print(person)
```

```
Apples
People
```

Organizing code

```
class Restaurant {
  let name: String

  var menuItems: [MenuItem]
  . . .
}

extension Restaurant {
  func add(menuItem: MenuItem)
  func remove(menuItem: MenuItem)

  // Other methods related to the Restaurant
}
```

Unit 2—Lesson 2
Lab: Extensions

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