

framework training
We love technology

Building iOS Apps in Swift

Course notes for part 1

14th-16th September 2021

020 3137 3920

@FrameworkTrain

frameworktraining.co.uk

Contents List

Day 1 - 14th September 2021

Session 1: Introduction to Swift	1
• Welcome and Introductions	1
• Introduction to Swift and Playgrounds	5
• Summary of Swift language basics	10
Session 2: Introduction to using Xcode	12
Session 3: UIKit: Views and controls	15
Session 4: Functions	26
Session 5: Prototyping an interface design	34
• Conference Attendee App Overview	37
• Making prototype app	39

Day 2 - 15th September 2021

Session 6: Structs	50
Session 7: Classes	67
Session 8: Working with AutoLayout and Stack layout	78
Session 9: Dictionaries/Optionals / Guard / Scope	90
• Dictionaries	90
• Optionals	94
• Guard	100
• Understanding scope of variables	103
Session 10: Advanced navigation in UIKit	109
• Segues and Navigation	109
• Tab bars	117
• Lifetime of an app	123

Day 3 - 16th September 2021

Session 11: Building dynamic tables	129
• First Animal App	130
• Second Animal App	135
• Third Animal App	140
• Fourth Animal App	144

Session 12: Protocols, closures	149
• Protocols	149
• Closures	161
Session 13: Location facilities	174
Session 14: Interacting with taps and the accelerometer	180
• Gestures	180
• Using the accelerometer	182

Session 1: Introduction to Swift

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

1

Welcome & Introductions

Chris Price

cjp@aber.ac.uk



Building iOS Apps in Swift

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

29/04/2021

Course Overview

3

Introductions - me



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

29/04/2021

Course Overview
2

4

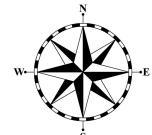
Introductions - you



□ Course attendees

- You...
 - Name
 - Professional Position
 - Programming experience
 - Motivation and Expectations

Course Content

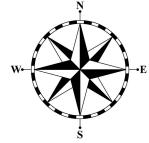


□ Part 1: First three days

- Basics of Swift language (briefly)
- Extra features of Swift needed for app building
- Enough about building iOS apps to build *most* self-contained apps

□ Part 2: Second three days - more advanced apps

- Persistence
- Linking to internet systems, cloud services, libraries
- Notifications
- Reactive systems and concurrency
- Automated testing



How will the course work

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

Questions/comments?



Introduction to Swift and Playgrounds

Chris Price



July 2021

1

Swift - a modern language: Safe

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

Swift - a modern language: Fast

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

3

Swift - a modern language: Expressive

- Doesn't make people write stuff the compiler should know:
- Implied type declaration where possible
- Implicit type name when type known (e.g. for enums)
- Has the features you might expect in a modern language
- Powerful Collections
- Protocols
- Extensions
- Functional programming
- Ints, Doubles etc are first class items

6

4

Hello, world

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

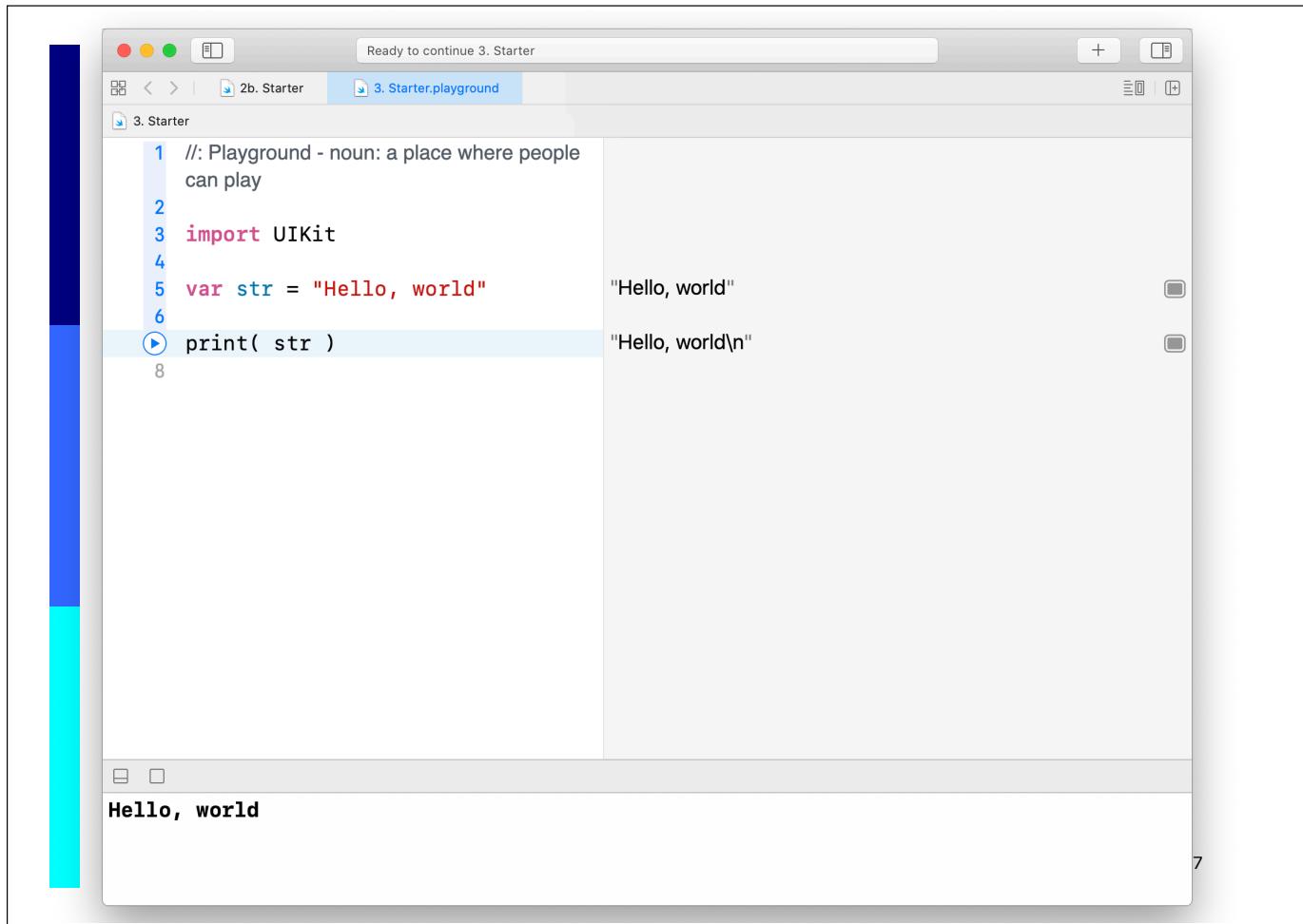
5

Try making a playground

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

7

6



A screenshot of the Xcode Playgrounds interface. The title bar says "Ready to continue 3. Starter". There are two tabs: "2b. Starter" and "3. Starter.playground", with "3. Starter.playground" being the active one. The code editor shows the following Swift code:

```
1 //: Playground - noun: a place where people  
can play  
2  
3 import UIKit  
4  
5 var str = "Hello, world"  
6  
7 print( str )  
8
```

The output pane shows the results of the code execution:

```
Hello, world
```

Explore playgrounds and language basics

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

More on material in this session

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

9

We will look in more detail at:

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

9

10

The Swift Language - Brief Overview (1)

Chris Price v1.4, Sept 2019

Simple Variable Types

Int, Double, Bool

Can be implicitly declared:

```
let x = 42 // Int constant
var y = 3.1 // Double variable
```

Explicit declaration:

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

Common Operators

Arithmetic Operators

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

Arithmetic shorthand

// e.g. y+=21	same as y = y+21
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Remainder and assign

Unary operators

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

Boolean tests - return true or false

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

Conditionals

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

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

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

Switch Statement

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

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

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

The Swift Language - Brief Overview (2)

Chris Price v1.4, Sept 2019

Strings

Simple string expressions and printing out

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

String interpolation

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

String tests

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

Equality tests whether made up of same characters in same order

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

Useful string features

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

Manipulating characters in a string

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

Arrays

//Declare empty array of strings

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

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

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

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

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

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

For Loops

for item in range { statements }

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

for item in collection { statements }

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

Equivalent to 'for item in collection' using range

```
for i in 0..

```

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

While Loops

while condition { statements }

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

The Swift Language - Brief Overview (3)

Chris Price v1.4, Sept 2019

Functions

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

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

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

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

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

Structures

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

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

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

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

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

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

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

Classes

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

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

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

    func printName(){
        print(fullName)
    }
}

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

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

The Swift Language - Brief Overview (4)

Chris Price v1.4, Sept 2019

Class inheritance

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

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

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

    func seePatient(patientName: String) {
        print("Hullo, " + patientName
              + ", I'm Dr " + fullName)
    }
}

let doc = Doctor(fName: "Jane",
                 lName: "Jones", spec: "Pediatrics")
doc.printName()
doc.seePatient(patientName: "George")
```

Enumerations

```
enum TempType {
    case degF
    case degC
}

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

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

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

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

Dictionaries

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

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

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

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

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

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

Session 2: Introduction to using Xcode

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

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

1

How do you make a multi-screen app?

Case Study: Crossflow Energy

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



Turbine

Crossflow SMART TURBINE



PV Array

Flexi or Fixed Solar pV with ground and tower mounting options



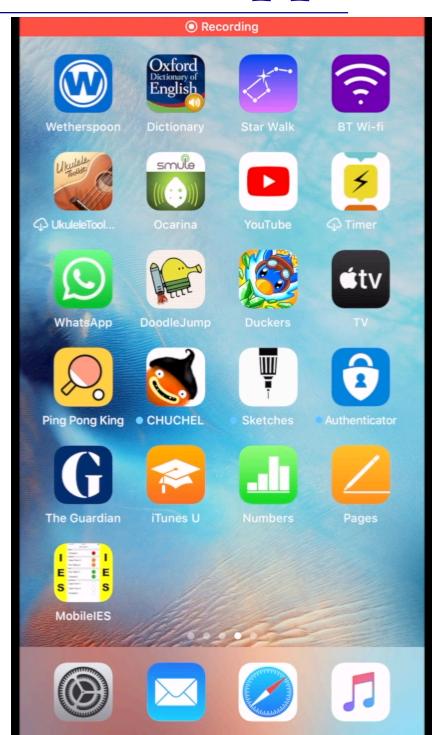
Battery

Latest rack mounted battery technology providing consistent power supply

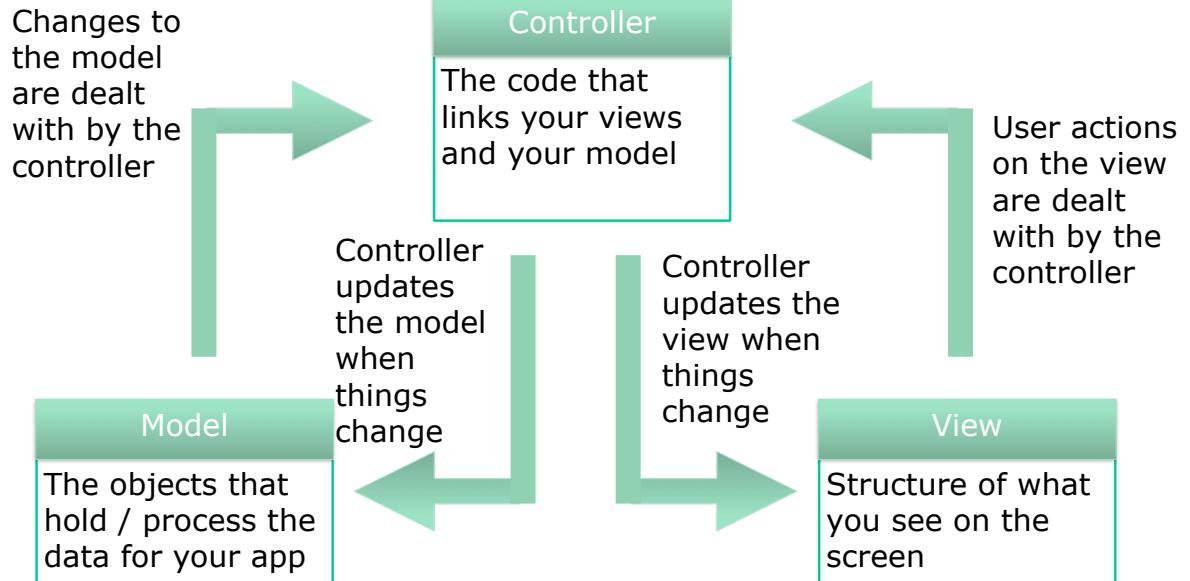


Generation or Grid

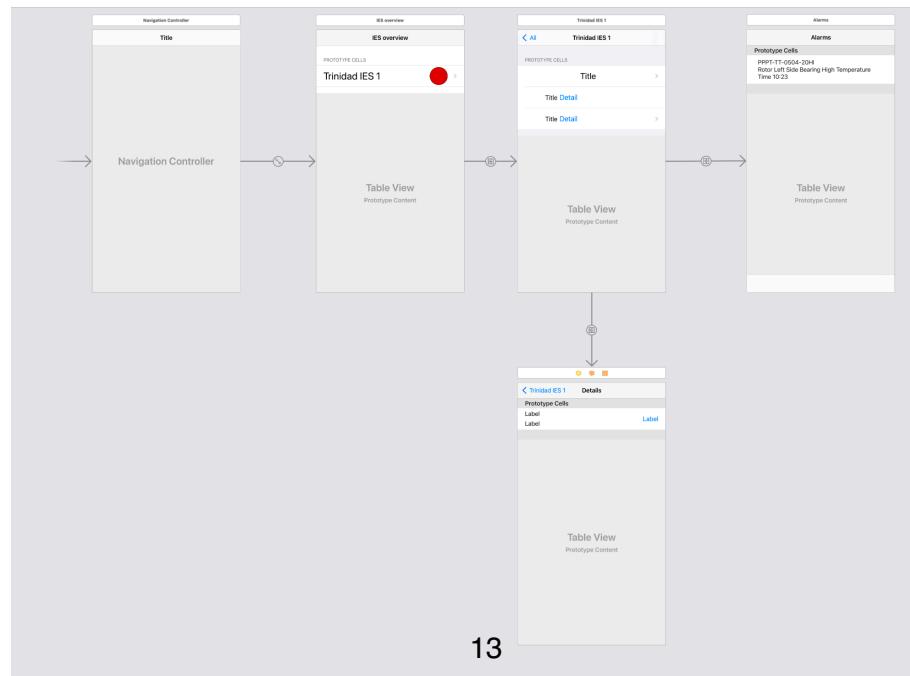
Other Energy Generation inputs optional



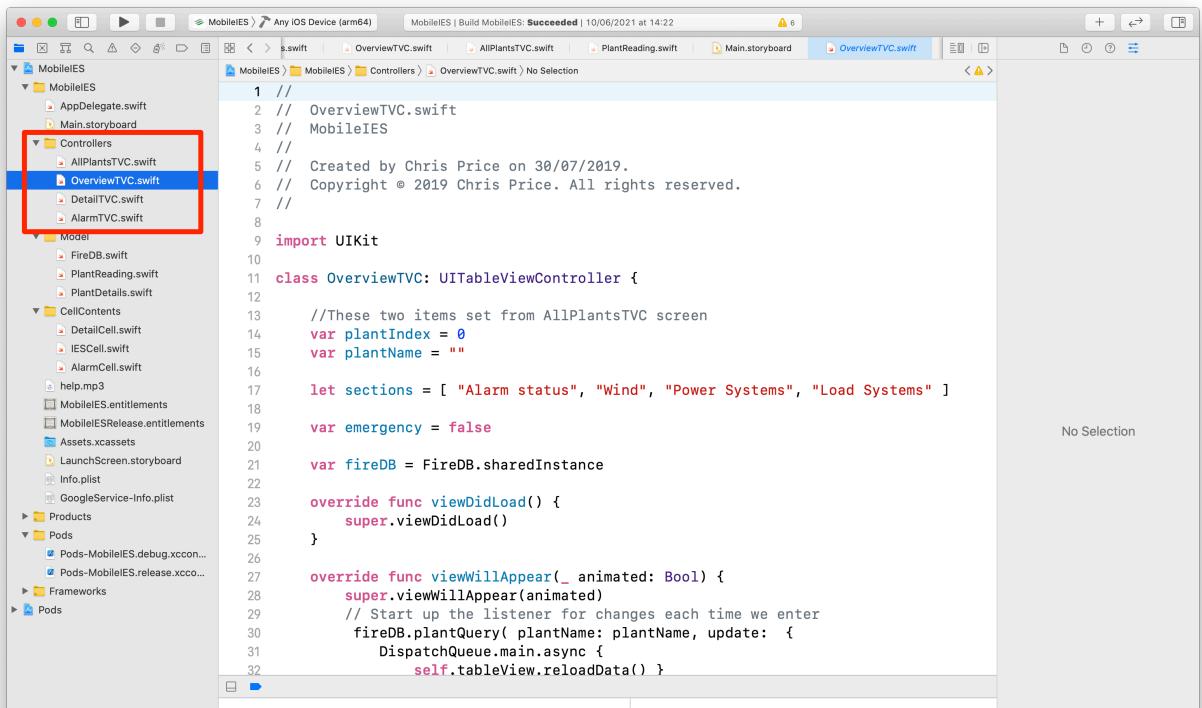
Model-View-Controller in Xcode



Views are made in Interface Builder in simple case, one view for each screen



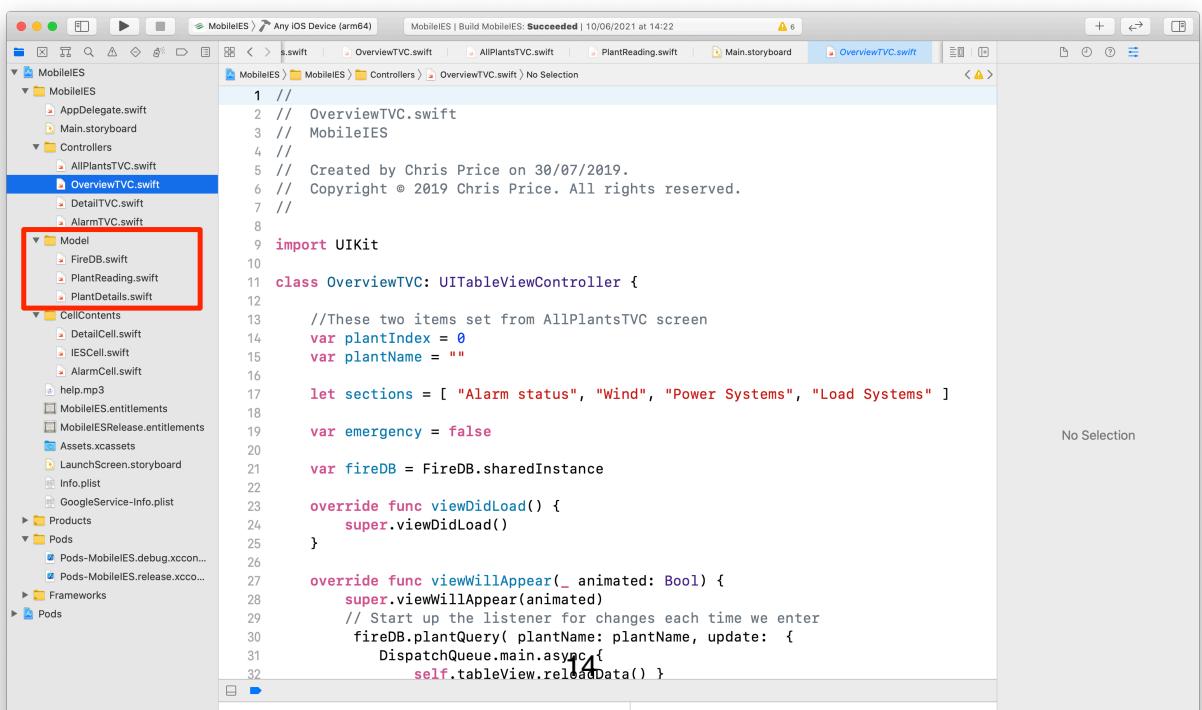
There will be view controller code for each of those screens



The screenshot shows the Xcode interface with the project 'MobileIES' open. The left sidebar displays the project structure, including 'MobileIES', 'MobileES', 'Controllers' (containing 'AllPlantsTVC.swift', 'OverviewTVC.swift', 'DetailTVC.swift', and 'AlarmTVC.swift'), 'Model' (containing 'FireDB.swift', 'PlantReading.swift', and 'PlantDetails.swift'), and various support files like 'Main.storyboard', 'Info.plist', and 'LaunchScreen.storyboard'. The right pane shows the code for 'OverviewTVC.swift'. A red box highlights the 'Controllers' folder and its contents.

```
1 //  
2 // OverviewTVC.swift  
3 // MobileIES  
4 //  
5 // Created by Chris Price on 30/07/2019.  
6 // Copyright © 2019 Chris Price. All rights reserved.  
7 //  
8  
9 import UIKit  
10  
11 class OverviewTVC: UITableViewController {  
12  
13     //These two items set from AllPlantsTVC screen  
14     var plantIndex = 0  
15     var plantName = ""  
16  
17     let sections = [ "Alarm status", "Wind", "Power Systems", "Load Systems" ]  
18  
19     var emergency = false  
20  
21     var fireDB = FireDB.sharedInstance  
22  
23     override func viewDidLoad() {  
24         super.viewDidLoad()  
25     }  
26  
27     override func viewWillAppear(_ animated: Bool) {  
28         super.viewWillAppear(animated)  
29         // Start up the listener for changes each time we enter  
30         fireDB.plantQuery( plantName: plantName, update: {  
31             DispatchQueue.main.async {  
32                 self.tableView.reloadData()  
33             }  
34         })  
35     }  
36  
37 }
```

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



The screenshot shows the Xcode interface with the project 'MobileIES' open. The left sidebar displays the project structure, including 'MobileIES', 'MobileES', 'Controllers' (containing 'AllPlantsTVC.swift', 'OverviewTVC.swift', 'DetailTVC.swift', and 'AlarmTVC.swift'), 'Model' (containing 'FireDB.swift', 'PlantReading.swift', and 'PlantDetails.swift'), and various support files like 'Main.storyboard', 'Info.plist', and 'LaunchScreen.storyboard'. The right pane shows the code for 'OverviewTVC.swift'. A red box highlights the 'Model' folder and its contents.

```
1 //  
2 // OverviewTVC.swift  
3 // MobileIES  
4 //  
5 // Created by Chris Price on 30/07/2019.  
6 // Copyright © 2019 Chris Price. All rights reserved.  
7 //  
8  
9 import UIKit  
10  
11 class OverviewTVC: UITableViewController {  
12  
13     //These two items set from AllPlantsTVC screen  
14     var plantIndex = 0  
15     var plantName = ""  
16  
17     let sections = [ "Alarm status", "Wind", "Power Systems", "Load Systems" ]  
18  
19     var emergency = false  
20  
21     var fireDB = FireDB.sharedInstance  
22  
23     override func viewDidLoad() {  
24         super.viewDidLoad()  
25     }  
26  
27     override func viewWillAppear(_ animated: Bool) {  
28         super.viewWillAppear(animated)  
29         // Start up the listener for changes each time we enter  
30         fireDB.plantQuery( plantName: plantName, update: {  
31             DispatchQueue.main.async {  
32                 self.tableView.reloadData()  
33             }  
34         })  
35     }  
36  
37 }
```



Session 3: UIKit: Views and controls

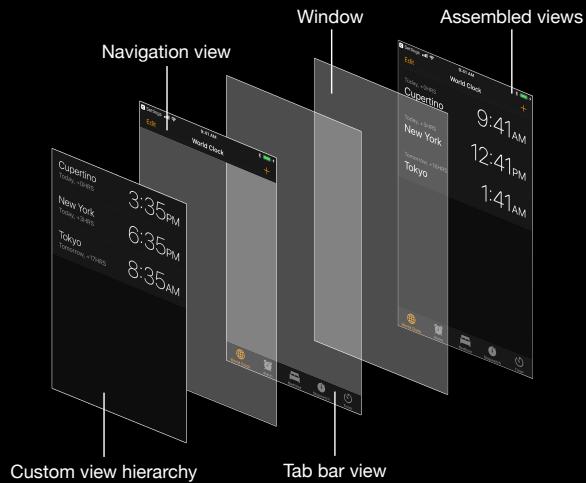
- Attributes of common views
- Adding controls to your app
- Adding more views
- *Lab 3: Building apps with more views and controls*

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

1

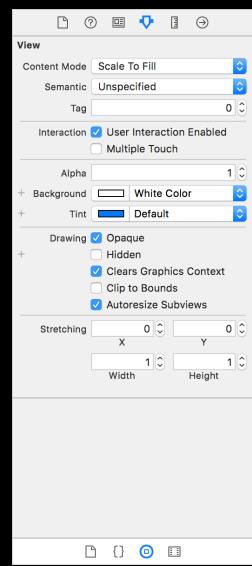
Unit 2—Lesson 7: Introduction to UIKit (p227 in book)

Common system views



Common system views Configuration

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



Label (UILabel)

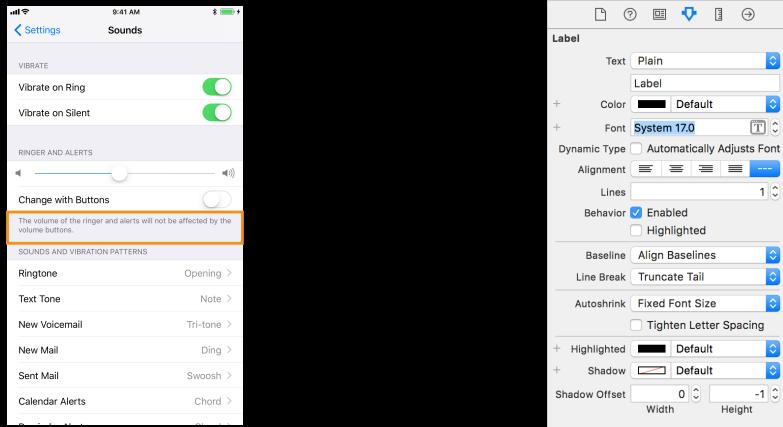
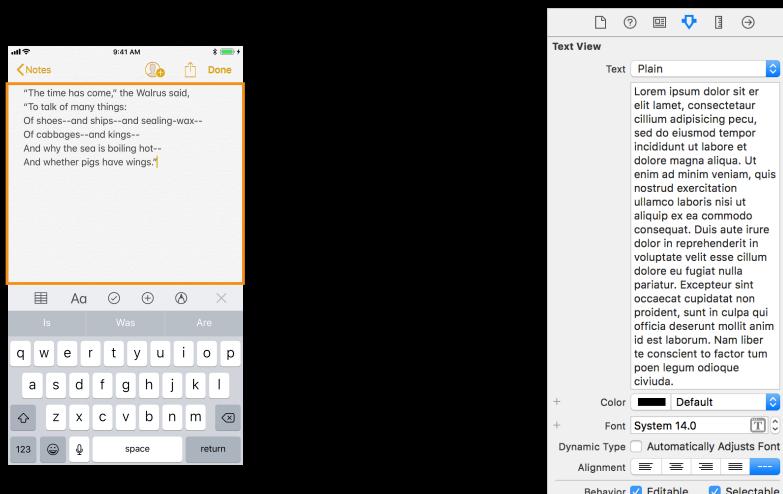


Image view (UIImageView)



Text view (UITextView)



Scroll view (UIScrollView)

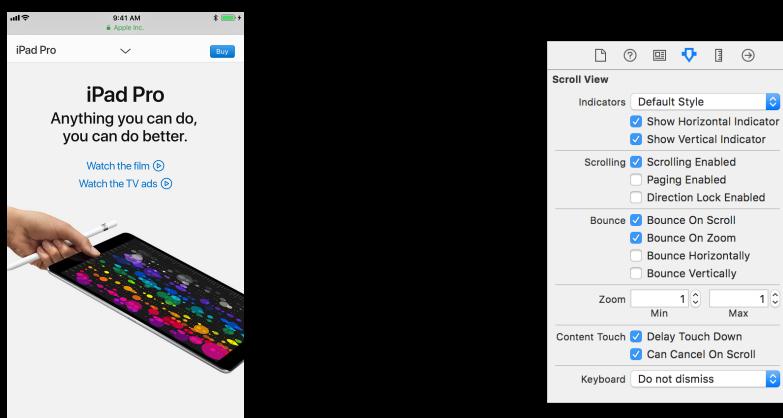
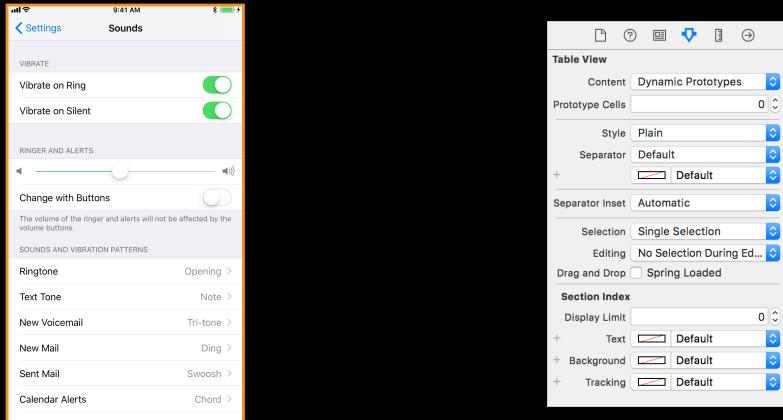
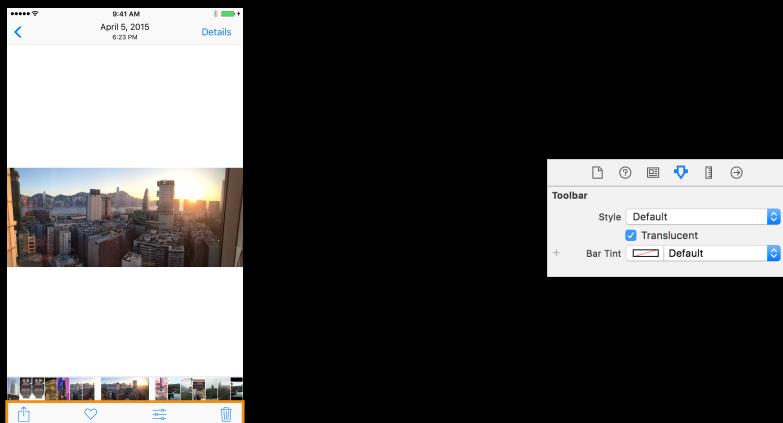


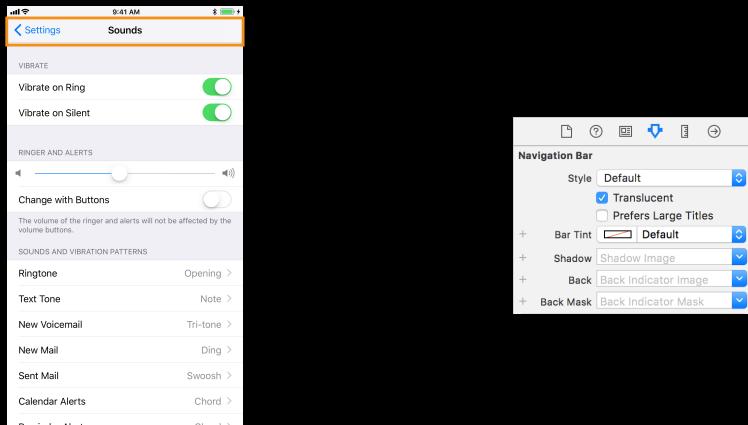
Table view (UITableView)



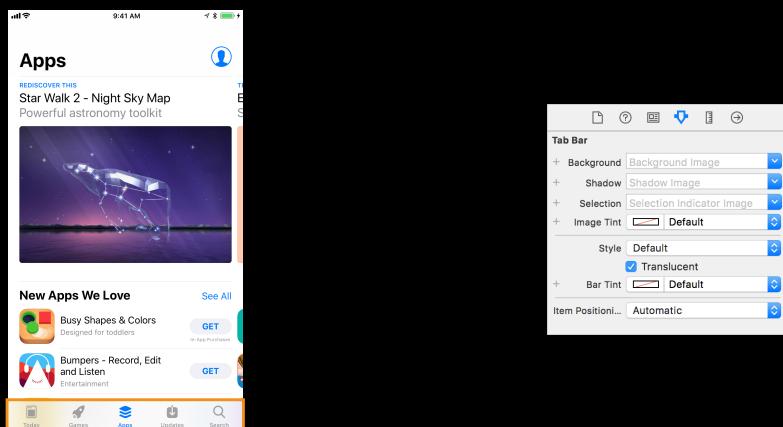
Toolbars (UIToolbar)



Navigation bars (UINavigationBar)

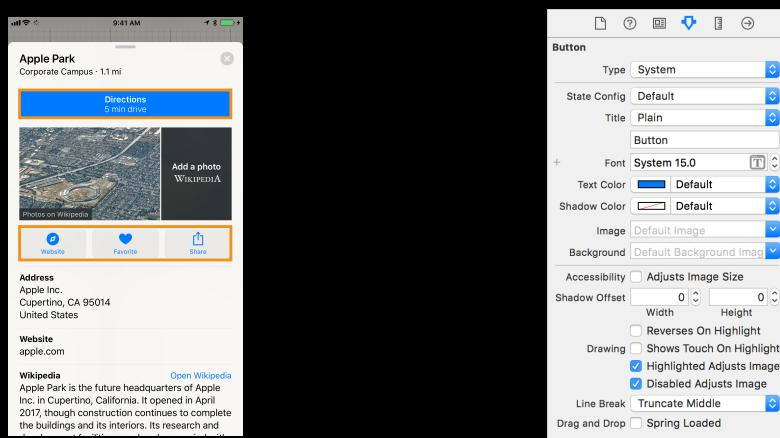


Tab bars (UITabBarController)

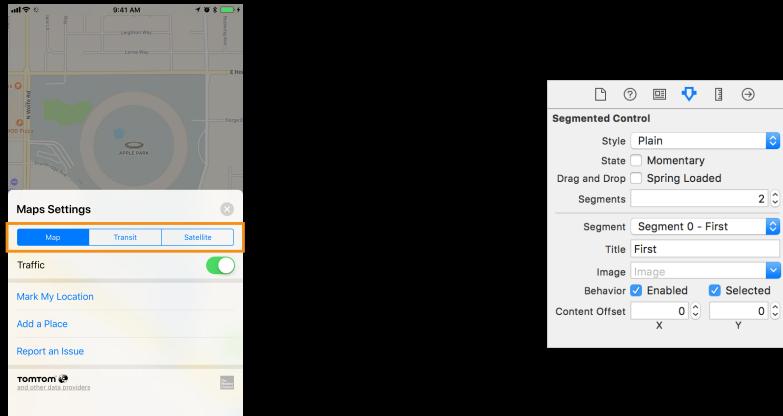


Controls

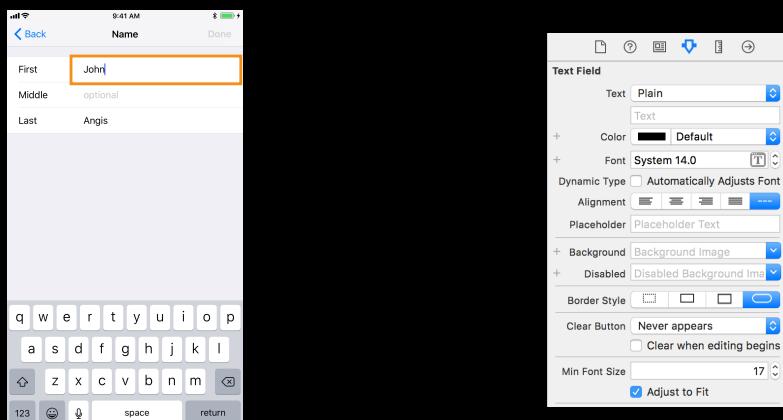
Buttons (UIButton)



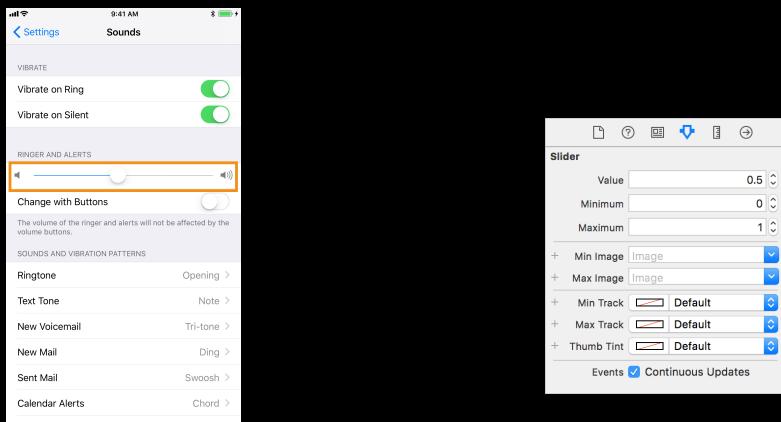
Segmented controls (UISegmentedControl)



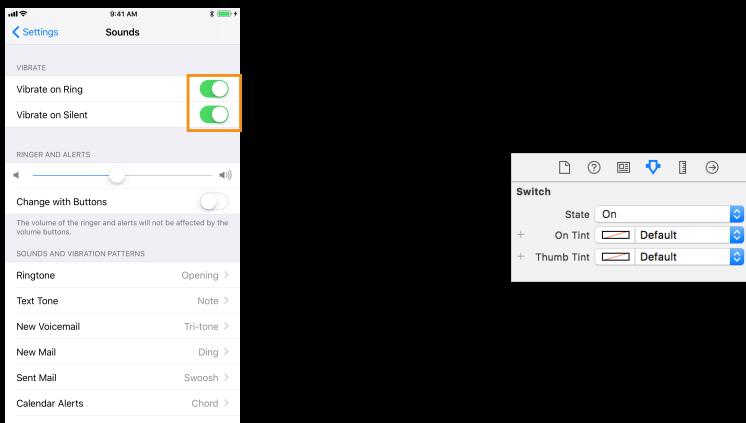
Text fields (UITextField)



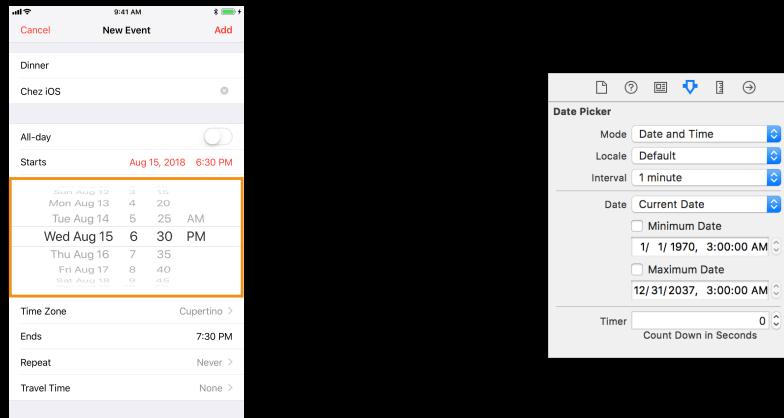
Sliders (UISlider)



Switches (UISwitch)



Date pickers (UIDatePicker)



UIKit User Interface Catalog

The image is a screenshot of the UIKit User Interface Catalog website, viewed in a web browser. The page title is 'UIKit User Interface Catalog'. On the left, there is a sidebar with a 'Views' section containing links to various view types: Action Sheets, Activity Indicators, Alert Views, Collection Views, Image Views, Labels, Navigation Bars, Picker Views, Progress Views, Scroll Views, Search Bars, Tab Bars, Table Views, Text Views, Toolbars, and Web Views. The main content area has a heading 'About Views' with the following text:

Views are the building blocks for constructing your user interface. Rather than using one view to present your content, you are more likely to use several views, ranging from simple buttons and text labels to more complex views such as table views, picker views, and scroll views. Each view represents a particular portion of your user interface and is generally optimized for a specific type of content. By building view upon view, you get a view hierarchy.

Below the text is a diagram illustrating the concept of 'Assembled views'. It shows a 'Window' containing a 'Navigation view', which in turn contains a 'World Clock' view. The 'World Clock' view displays time zones for Cupertino, New York, and Moscow. The entire assembly is labeled 'Assembled views'.



Practical



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

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



Session 4: Functions

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

- This covers lesson 2.2 of Development in Swift Fundamentals

1

Unit 2—Lesson 2: Functions

Functions

```
tieMyShoes()
```

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

Functions

Defining a function

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

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

```
3.1415926535
```

Parameters

```
func triple(value: Int) {  
    let result = value * 3  
    print("If you multiply \(value) by 3, you'll get \(result).")  
}  
  
triple(value: 10)
```

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

Parameters

Multiple parameters

```
func multiply(firstNumber: Int, secondNumber: Int) {  
    let result = firstNumber * secondNumber  
    print("The result is \(result).")  
}  
  
multiply(firstNumber: 10, secondNumber: 5)
```

```
The result is 50.
```

Return values

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

Return values

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

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

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

Argument labels

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

Argument labels

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