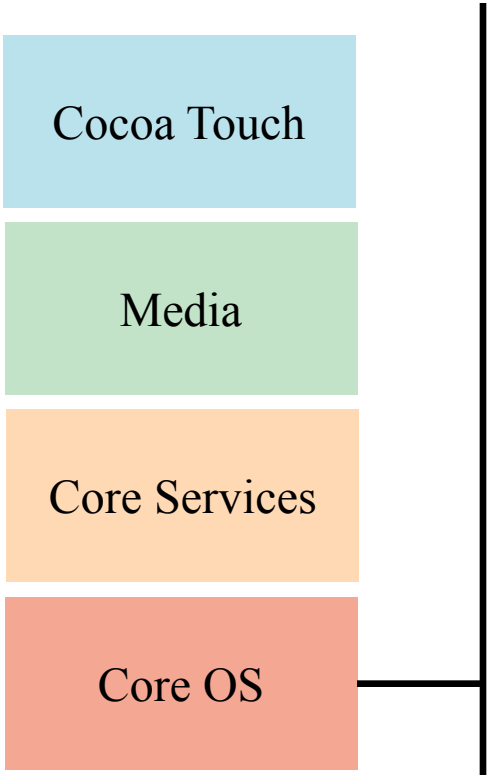
Mobile Design – IOS

Outline

- IOS Overview
- Swift Language
- Xcode Basics
- Design Strategy: Model-View-Controller (MVC)
- Multiple Views & View Controllers
- CocoaPods: Use External Dependencies
- References

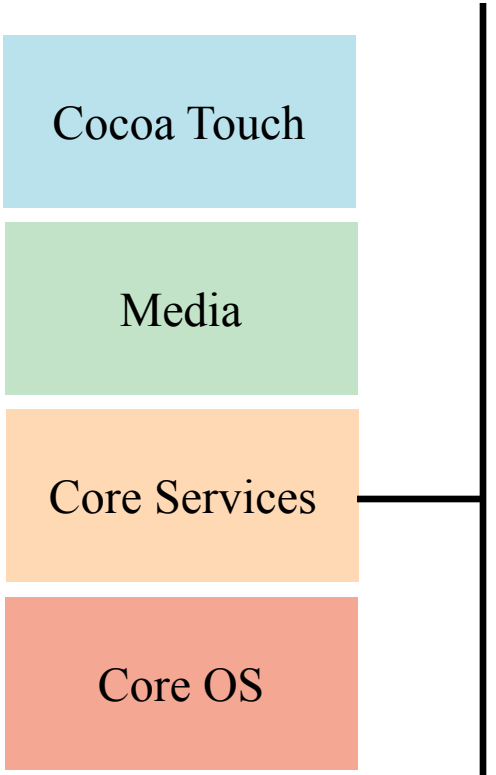
IOS Overview - What's in IOS?



Core OS

OSX Kernel	Power Management
Mach 3.0	Keychain Access
BSD	Certificates
Sockets	File System
Security	Bonjour

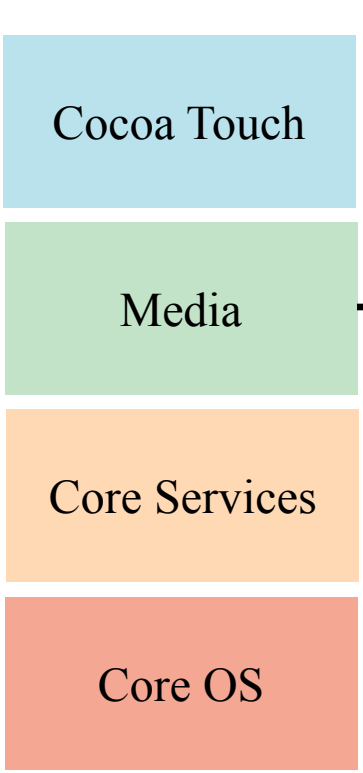
IOS Overview - What's in IOS?



Core Services

Collections	Core Location
Address Book	Net Services
Networking	Threading
File Access	Preferences
SQLite	URL Utilities

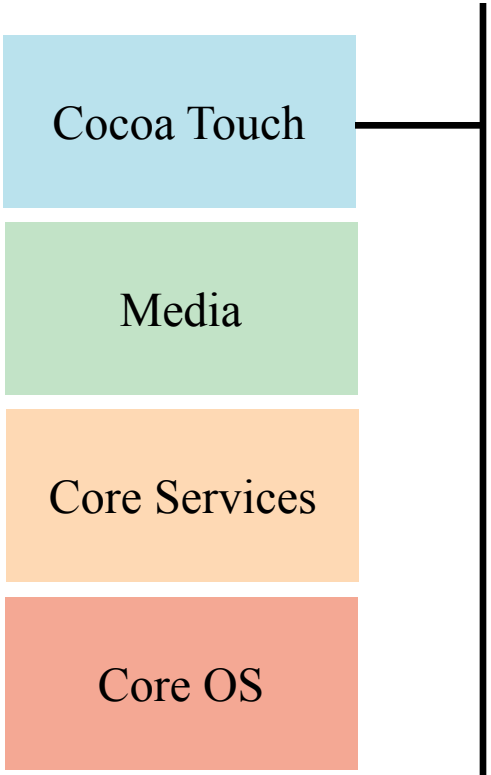
IOS Overview - What's in IOS?



Media

Core Audio	JPEG, PNG, TIFF
OpenAL	PDF
Audio Mixing	Quartz (2D)
Audio Recording	Core Animation
Video Playback	OpenGL ES

IOS Overview - What's in IOS?



Cocoa Touch

Multi-Touch	Alerts
Core Motion	Web View
View Hierarchy	Map Kit
Localization	Image Picker
Controls	Camera

IOS Platform Components

- Tools: Xcode, Instruments
- Language: Swift
- Frameworks: Foundation, Core Data, UIKit, Core Motion, Map Kit
- Design Strategy: MVC

Swift

- Swift Introduction
- Define simple values
- Control-flow: if-else
- Define a function
- Define a class
- Inherit a class

Introduction to Swift

Swift is a general-purpose, multi-paradigm, compiled programming language developed by Apple Inc. for iOS, macOS, watchOS, tvOS, and Linux.

•Swift is designed to work with Apple's Cocoa and Cocoa Touch frameworks and the large body of existing Objective-C (ObjC) code written for Apple products. It is built with the open source LLVM compiler framework and has been included in Xcode since version 6.

•Swift was introduced at Apple's 2014 Worldwide Developers Conference (WWDC).

•Version 2.2 was made open-source software under the Apache License 2.0 on December 3, 2015, for Apple's platforms and Linux.

•See: https://swift.org/

Introduction to Swift (cont'd)

Swift is friendly to new programmers. Swift removes the occurrence of large classes of common programming errors by adopting modern programming patterns:

- Variables are always initialized before use.
- Array indices are checked for out-of-bounds errors.
- Integers are checked for overflow.
- *Optionals* ensure that nil values are handled explicitly.
- Memory is managed automatically.
- Error handling allows controlled recovery from unexpected failures.

Swift – simple values

Use **let** to make a constant and **var** to make a variable.

```
var myVariable = 42
myVariable = 50
let myConstant = 42
let label = "The width is "
```

To include values in a string:

```
let apples = 3
let appleSummary = "I have \(apples) apples."
```

Most times the compiler infers the type of constant/variable for you. But sometimes you have to write the variable type explicitly:

```
let implicitInteger = 70
let explicitDouble: Double = 70
```

Swift – simple values (cont'd)

To create arrays and dictionaries:

```
var shoppingList = ["catfish", "water", "tulips", "blue
paint"]
shoppingList[1] = "bottle of water"

var occupations = [
    "Malcolm": "Captain",
    "Kaylee": "Mechanic",
]
occupations["Jayne"] = "Public Relations"
```

To create an empty array or dictionary, use the initializer syntax.

```
let emptyArray = [String]()
let emptyDictionary = [String: Float]()
```

Swift – Control Flow

Example: use **if** to make conditionals:

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
```

An optional value either contains a value or contains **nil** to indicate that a value is missing (append ? to any type).

```
var optionalName: String? = "John Appleseed"
if let name = optionalName {
    print("Hello, \(name)")   //name != nil
}
```

Swift – Define a function

Use func to declare a function. Call a function by following its name with a list of arguments in parentheses. Use -> to separate the parameter names and types from the function's return type.

```
func greet(person: String, day: String) ->  String  {
    return "Hello \(person), today is \(day)."
}
greet(person: "Bob", day: "Tuesday")
```

Swift – Define a class

Define a class:

```
class Shape {
    var numberOfSides = 0

    //called when an instance is created (Constructor)
    init(numberOfSides: Int) {
        self.numberOfSides = numberOfSides
    }


    func simpleDescription() -> String {
        return "A shape with \(numberOfSides) sides."
    }
}
```

Create a class instance:

```
let square = Shape(numberOfSides: 4)
square.simpleDescription()
```

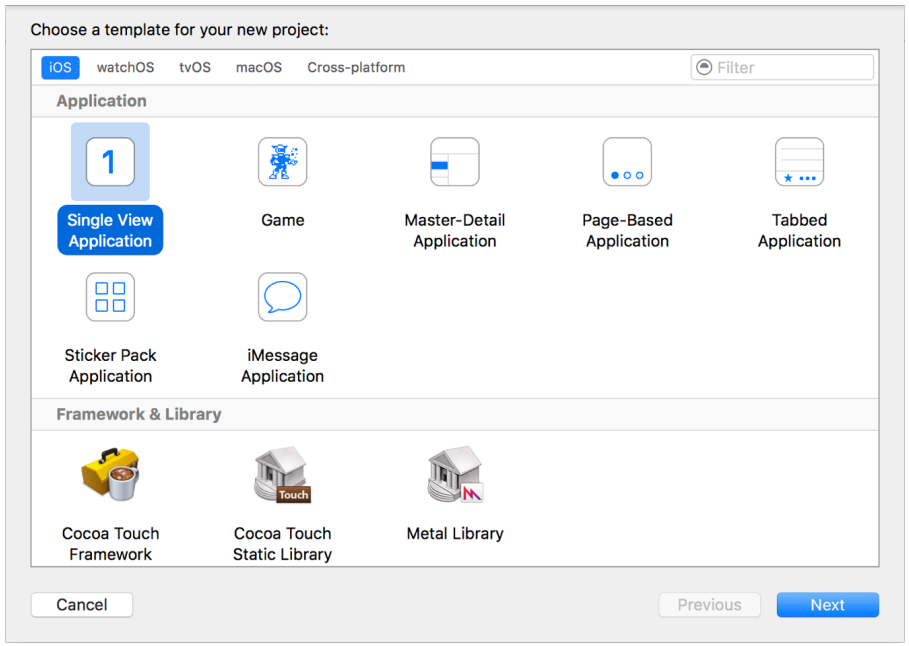
Swift – Inherit a class

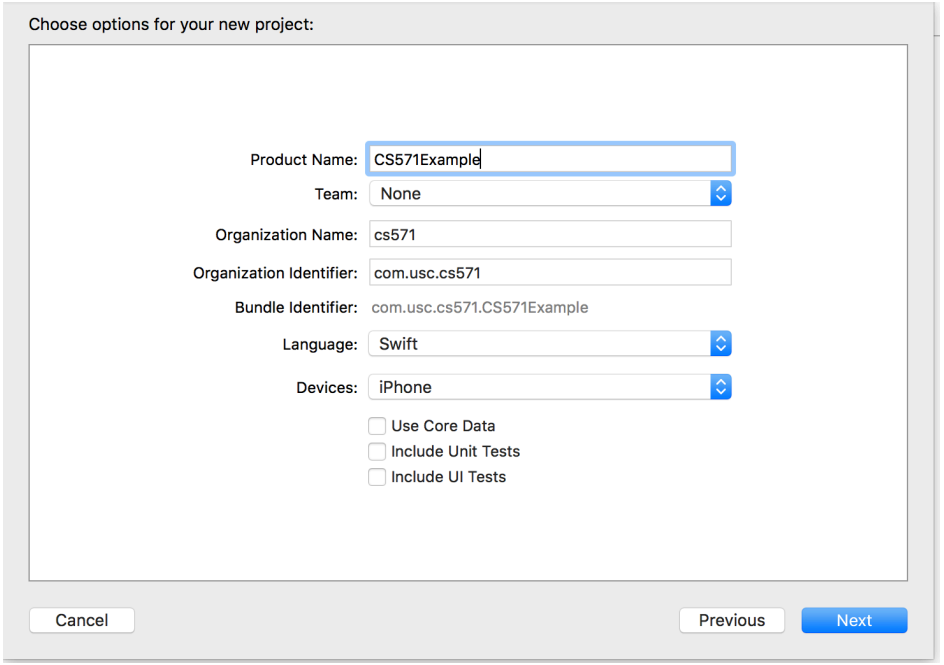
```
class Square: Shape {
    var sideLength: Double

    init(sideLength: Double, numberOfSides: Int) {
        self.sideLength = sideLength
        super.init(numberOfSides: numberOfSides)
    }

    func area() -> Double {
        return sideLength * sideLength
    }

    override func simpleDescription() -> String {
        return "A square with \(sideLength)."
    }
}
```

Xcode Basics

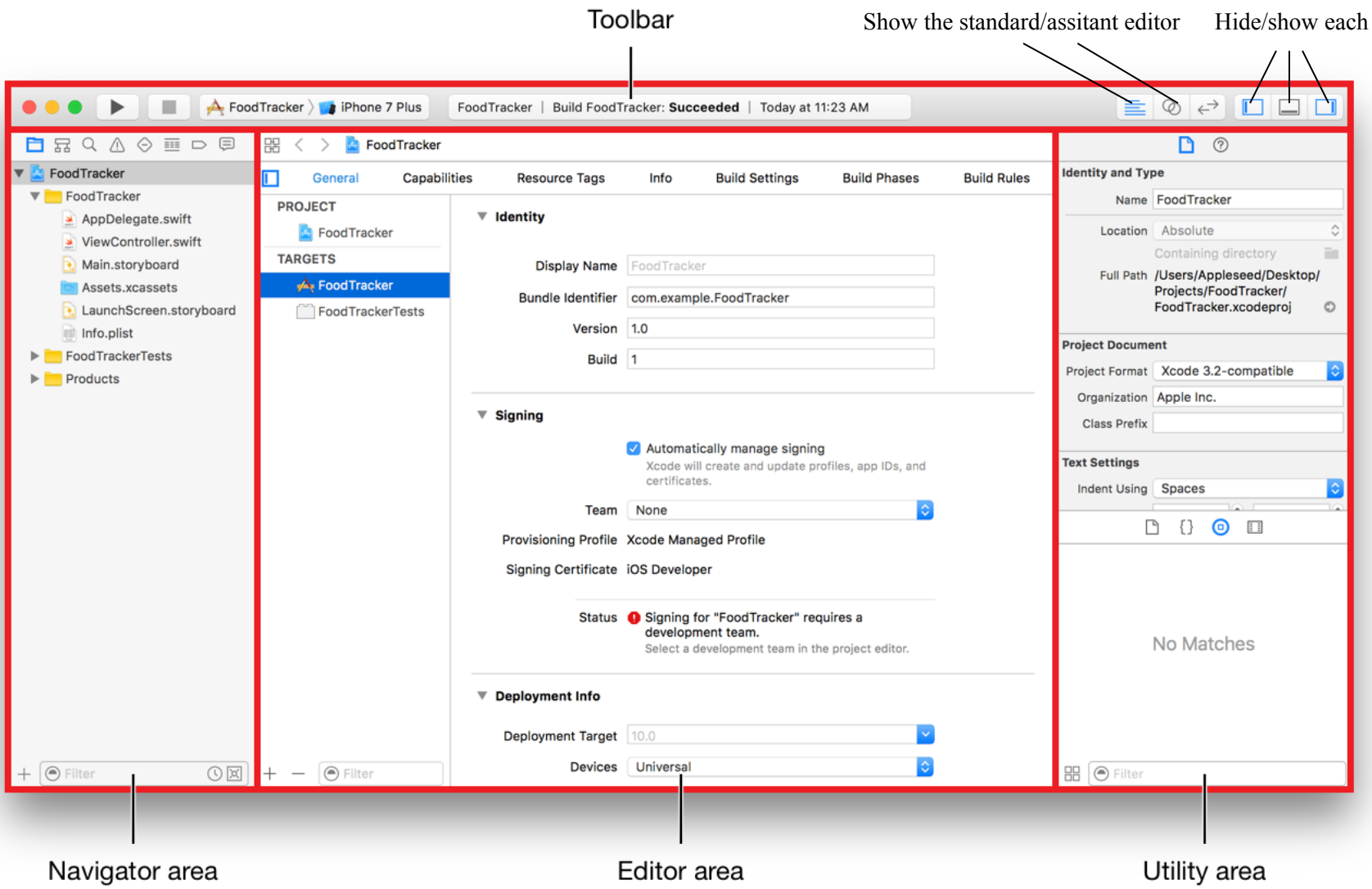
- Create a new project
- Get familiar with Xcode
- Design UI in storyboard
- Set view controller for the UI
- View controller lifecycle
- Connect UI to code
- Run your app in the simulator

Create a new project





Get familiar with Xcode

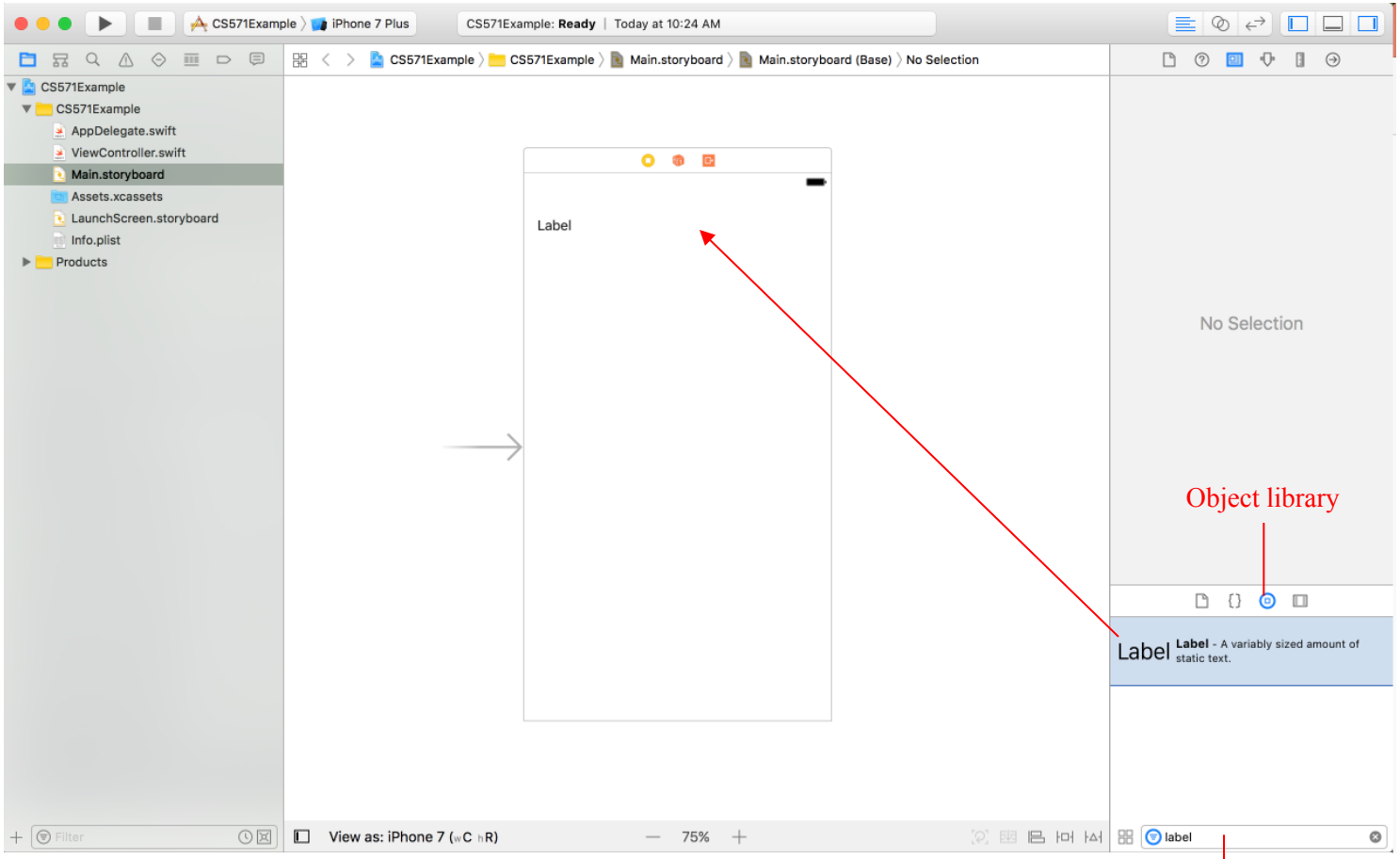


Storyboard

- A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens;
- A storyboard is composed of a sequence of scenes, each of which represents a view controller and its views;
- Scenes are connected by segue objects, which represent a transition between two view controllers.

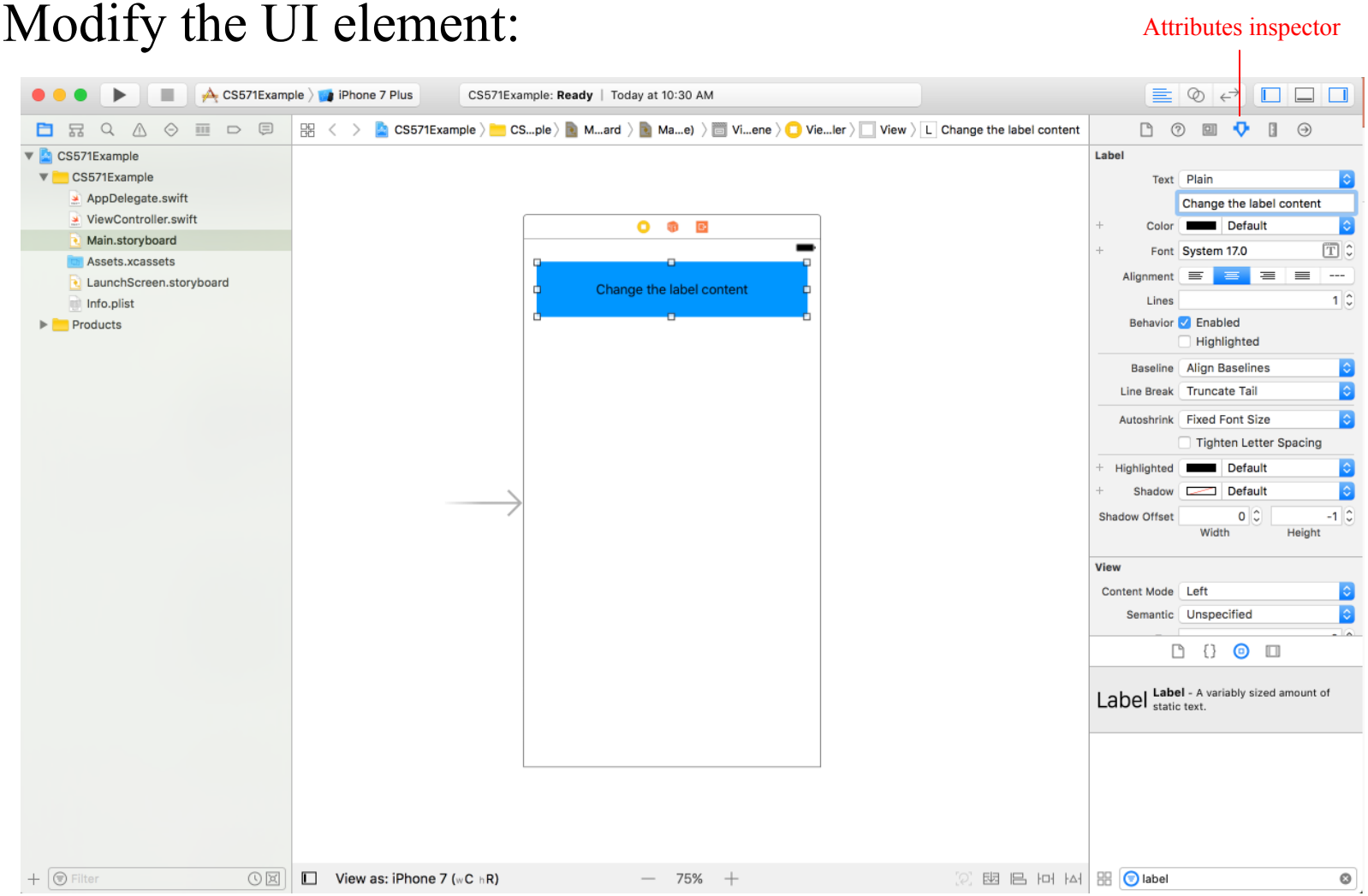
Design UI in storyboard

Add a UI Element to storyboard:



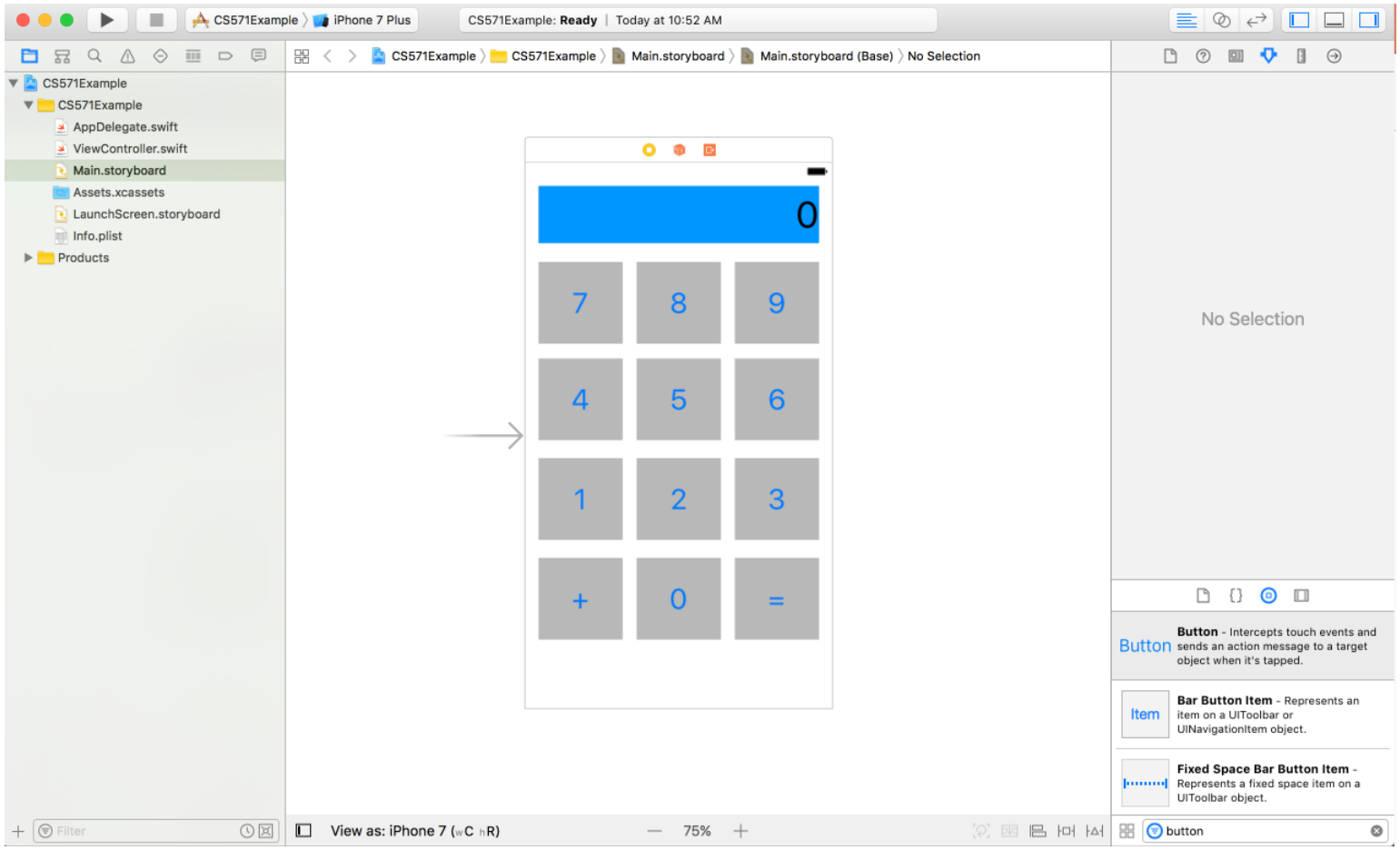
Design UI in storyboard (cont'd)

Modify the UI element:



Design UI in storyboard (cont'd)

Continue to add 12 buttons:



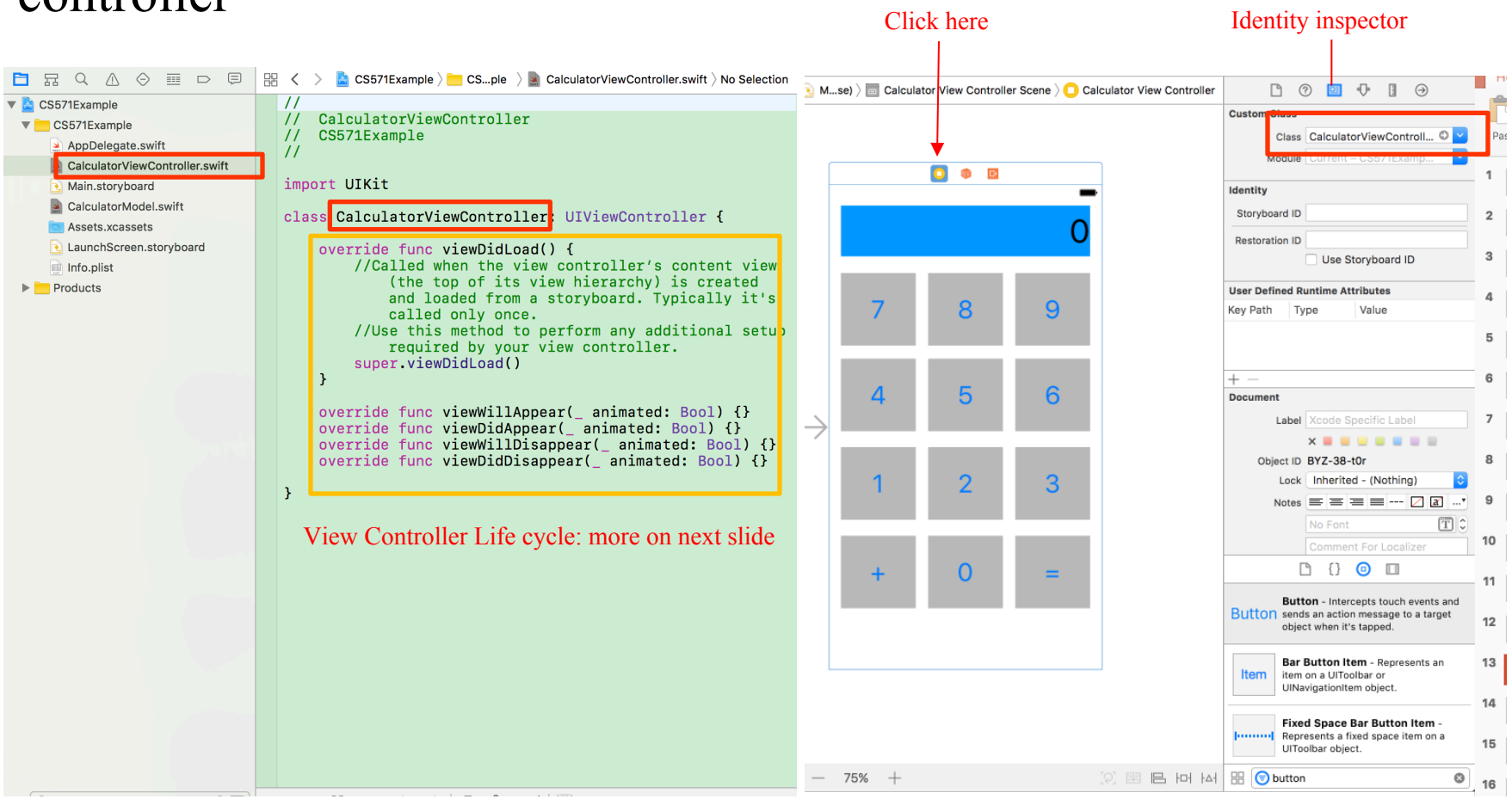
View Controller

Provides the infrastructure for managing the views of your UIKit app.

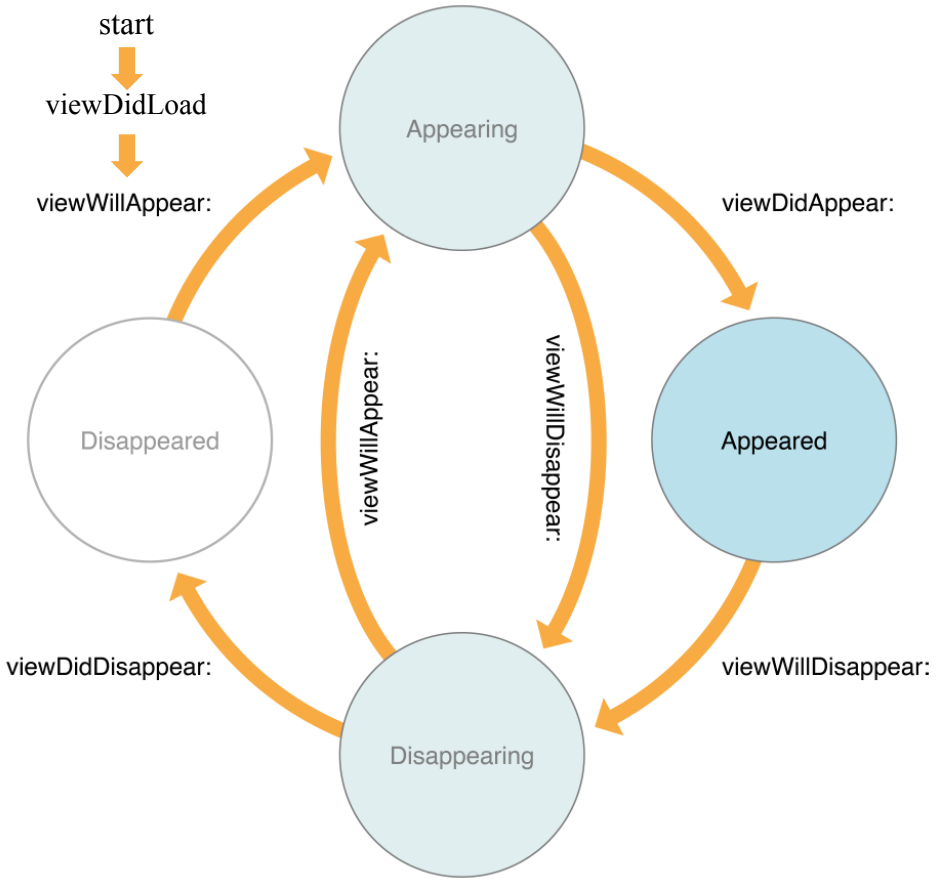
- A view controller manages a set of views that make up a portion of your app's user interface.
- It is responsible for loading and disposing of those views, for managing interactions with those views, and for coordinating responses with any appropriate data objects.
- View controllers also coordinate their efforts with other controller objects—including other view controllers—and help manage your app's overall interface.

Set View Controller for the UI

A common mistake for beginners is forgetting to set the view controller

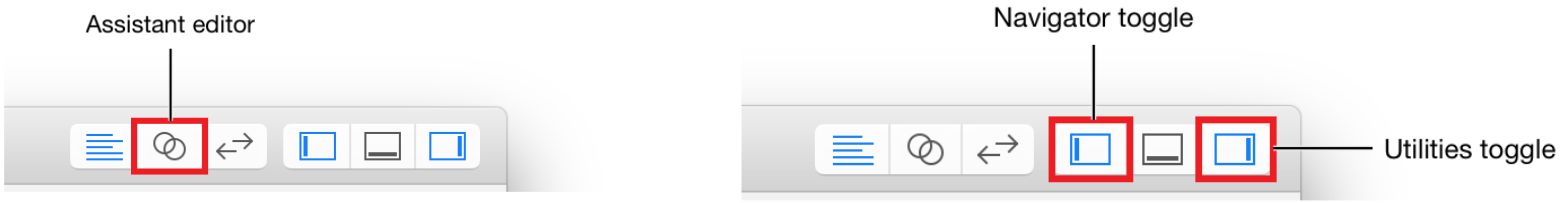


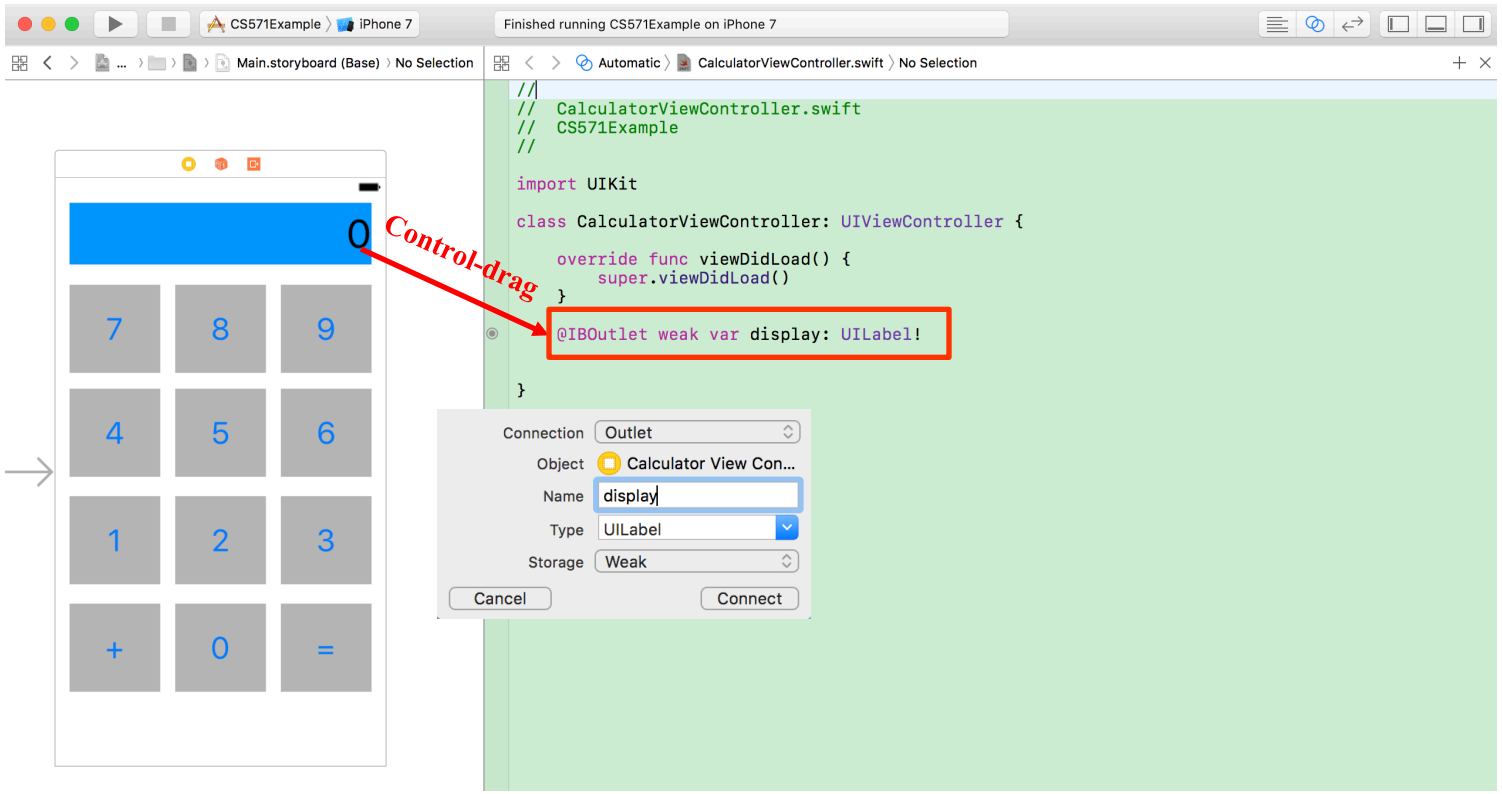
View Controller Lifecycle



An object of the `UIViewController` class (and its subclasses) comes with a set of methods that manage its view hierarchy. iOS automatically calls these methods at appropriate times when a view controller transitions between states.

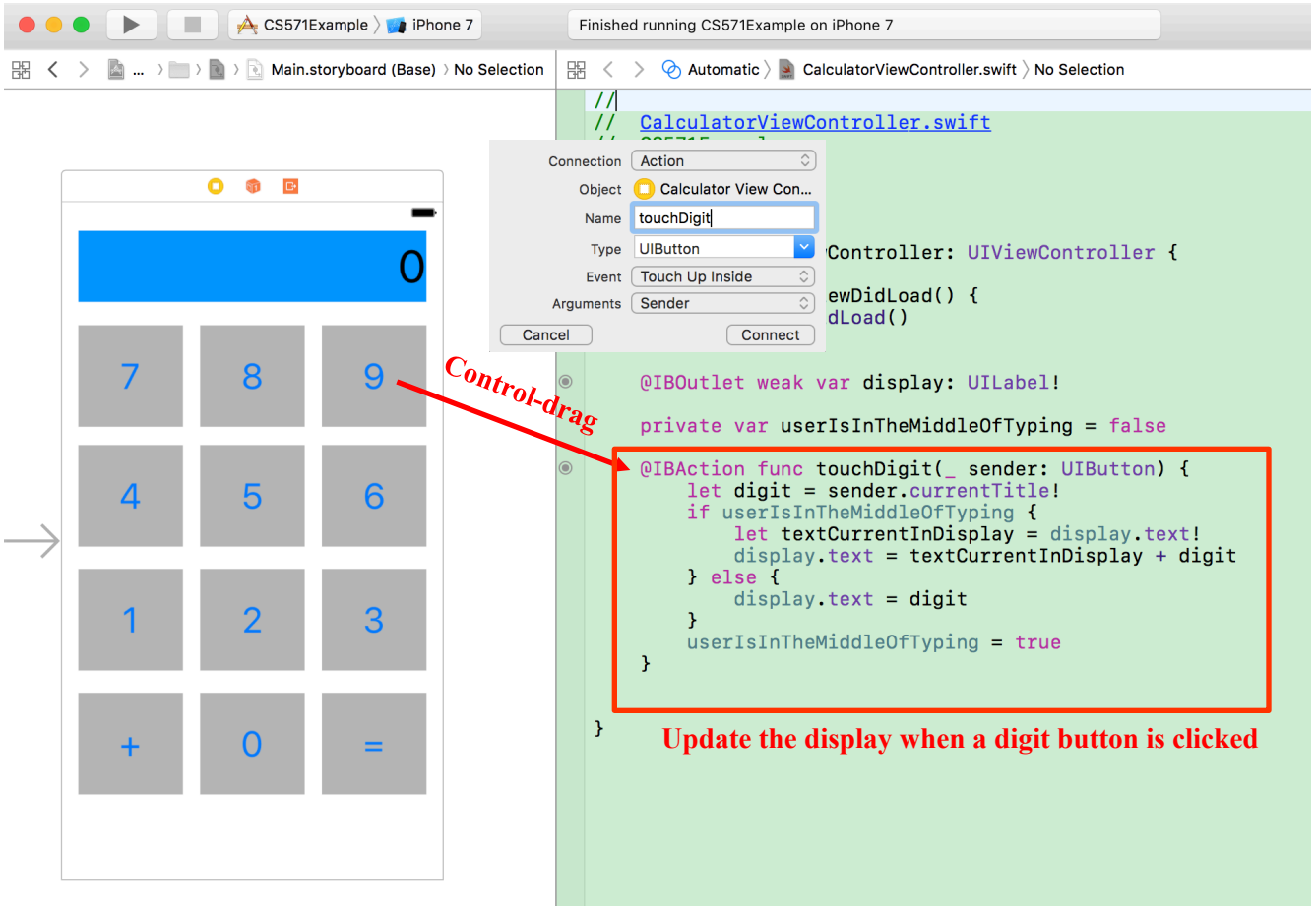
Connect UI to Code



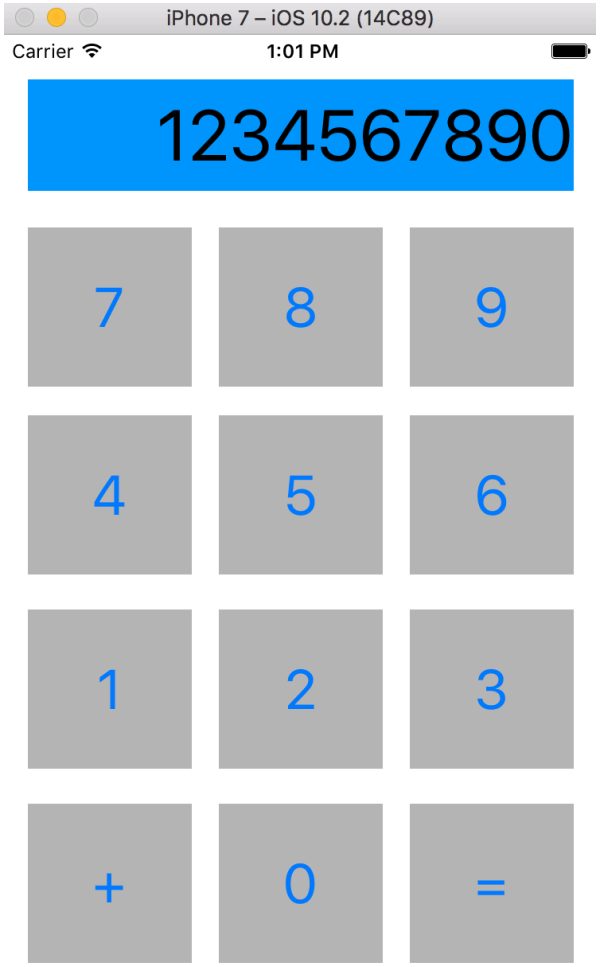


Connect UI to Code (cont'd)

Control-drag a digit button to create an event handler func. Then control-drag all the other digit buttons to the same func.



Run your app in the Simulator



- The Scheme pop-up menu lets you choose which simulator or device you'd like to run your app on.
- Click Run button.
- Click each of the digit buttons to test your app.

Design Strategy: Model-View-Controller (MVC)

- Why MVC
- How MVC works in IOS development
- Create a calculator model
- Design the UI view
- Controller: connect UI and the model

Why MVC?

MVC is central to a good design for a Cocoa application. The benefits of adopting this pattern are numerous.

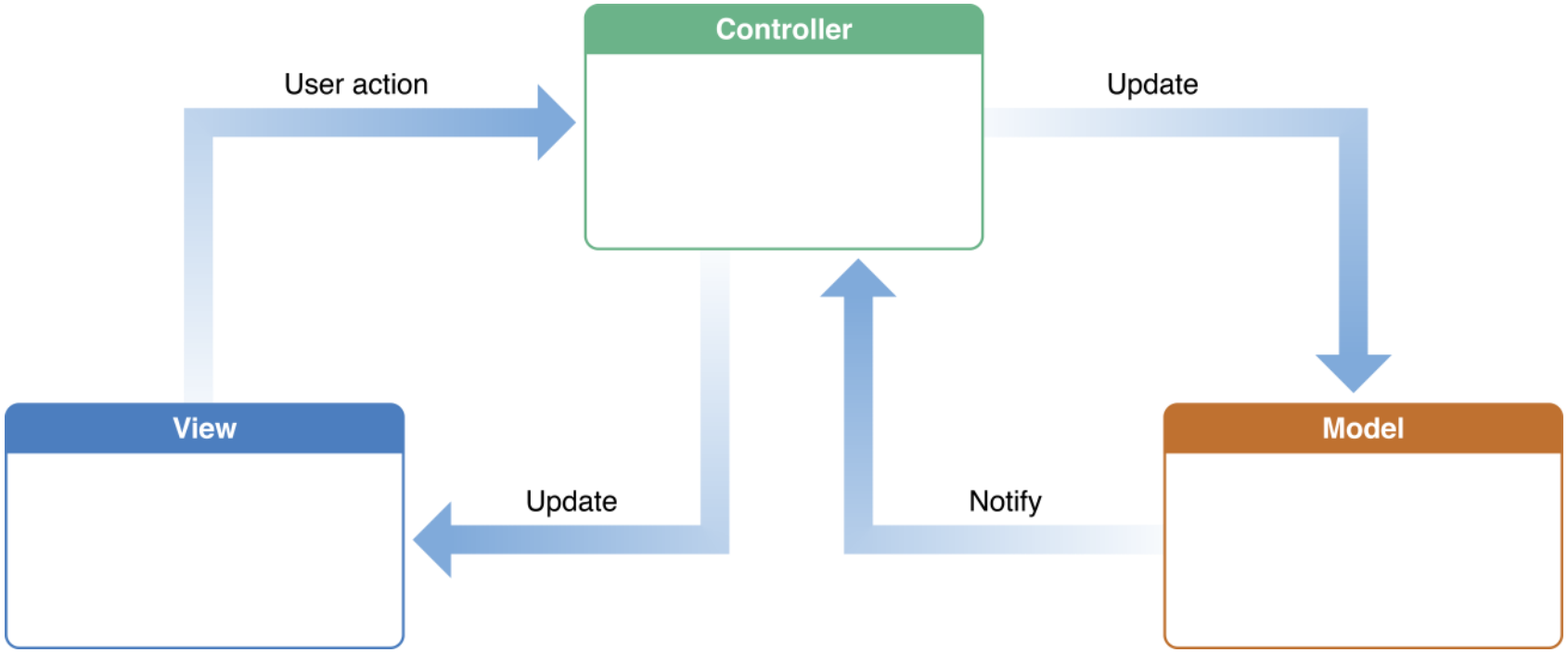
•Objects in the applications tend to be more reusable

•The interfaces tend to be better defined

•Applications having an MVC design are also more easily extensible than other applications.

•IOS development technologies and architectures are based on MVC and require that your custom objects play one of the MVC roles.

How MVC works in IOS development



Model = What your application is (but ***not*** how it is displayed)
Controller = How your Model is presented to the user (UI logic)
View = Your Controller's minions

Create a calculator model

What does the model do:

•Given the operands and operation symbols, return the result, such as 1+1=2

•Need to deal with 1+2+3+4=? and return any intermediate results when a "+" or "=" button is pressed.

•Perform a new operation:

- "+": Execute the pending operation to get intermediate result. Save the operation symbol and first operand (the intermediate result) as a pending operation.
- "=": Execute the pending operation to get final result.

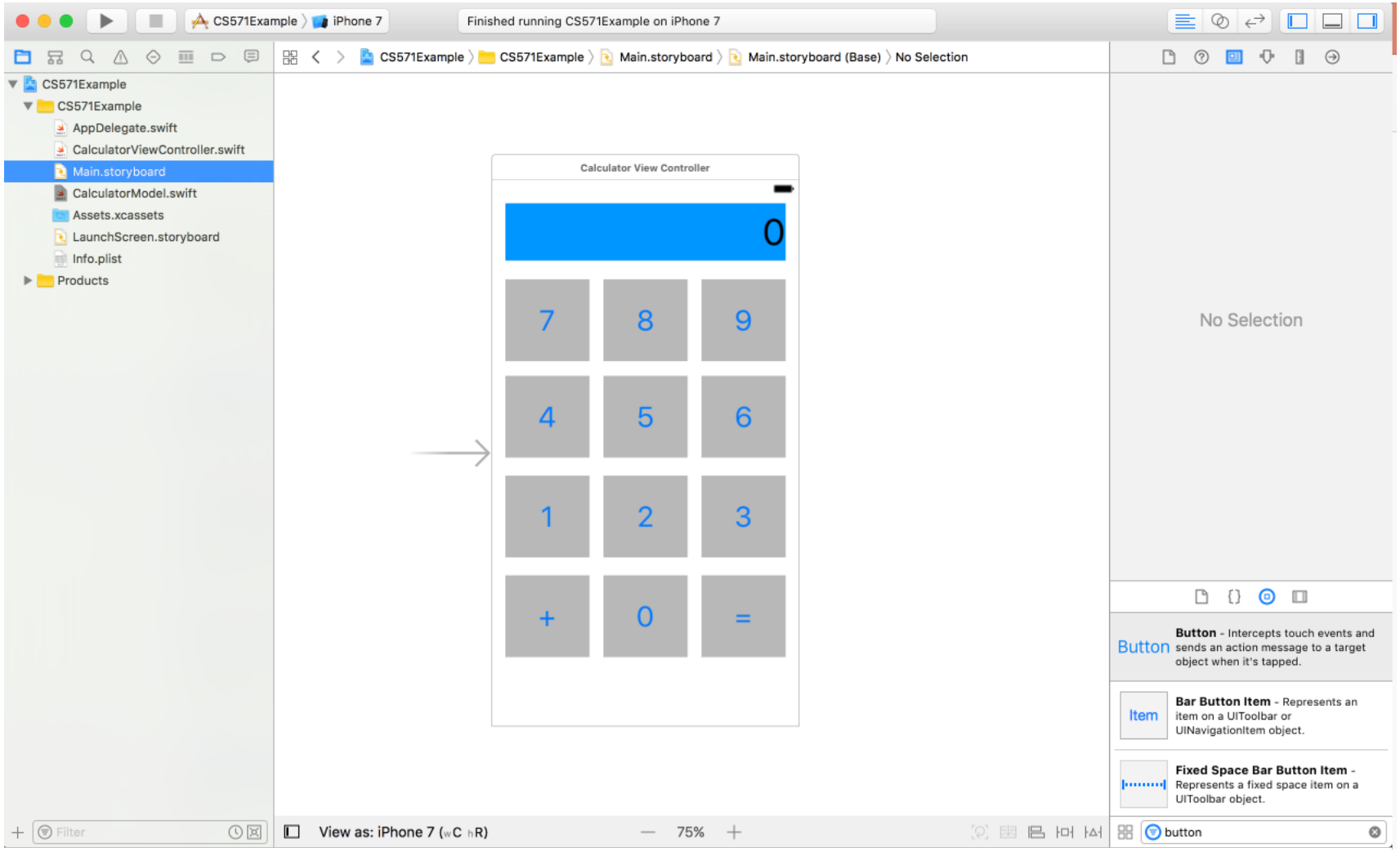
Create a calculator model (cont'd)

```
class CalculatorModel {
    //A dummy calculator model to support simple addition operation
    private var operations: Dictionary<String, Operation> = [
        "+": Operation.AdditionOperation({$0 + $1}),
        "=": Operation.Equal
    ]
    private enum Operation {
        case AdditionOperation((Int, Int) -> Int)
        case Equal
    }
    private struct PendingAdditionOperationInfo {
        var additionFunction: (Int, Int) -> Int
        var firstOperand: Int
    }

    private var accumulator = 0  //intemediate result
    private var pending: PendingAdditionOperationInfo?
    var result: Int { get { return accumulator } }

    func setOperand(operand: Int) {
        accumulator = operand
    }
```

Create a calculator model (cont'd)

```
    func performOperation(symbol: String) {
        if let operation = operations[symbol] {
            switch operation {
                case .AdditionOperation(let function):
                    executePendingAdditionOperation()
                    pending = PendingAdditionOperationInfo(additionFunction:
function, firstOperand: accumulator)
                case .Equal:
                    executePendingAdditionOperation()
            }
        }
    }

    private func executePendingAdditionOperation() {
        if pending != nil {
            accumulator = pending!.additionFunction(pending!.firstOperand,
accumulator)
            pending = nil
        }
    }

}
```

Design the UI view



Controller: connect UI and the model

What does the controller do:

•Get user actions from the UI view, let the model do the calculation, get results from model and update the UI view.

•Connection with the UI view:

- Own the outlet to the display label: can get and update the display
- Action handlers for all the digit buttons and operation symbol buttons

•Connection with the model:

- Send new operands and operation symbols to the model. Let model do the calculation.
- Get intermediate results and final results from the model

Controller: connect UI and the model (cont'd)

```
import UIKit

class CalculatorViewController: UIViewController {

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

    private var userIsInTheMiddleOfTyping = false

    private var displayValue: Int {
        get { return Int(display.text!)! }
        set { display.text = String(newValue) }
    }

    private var model = CalculatorModel()

    @IBOutlet weak var display: UILabel!
```

Controller: connect UI and the model (cont'd)

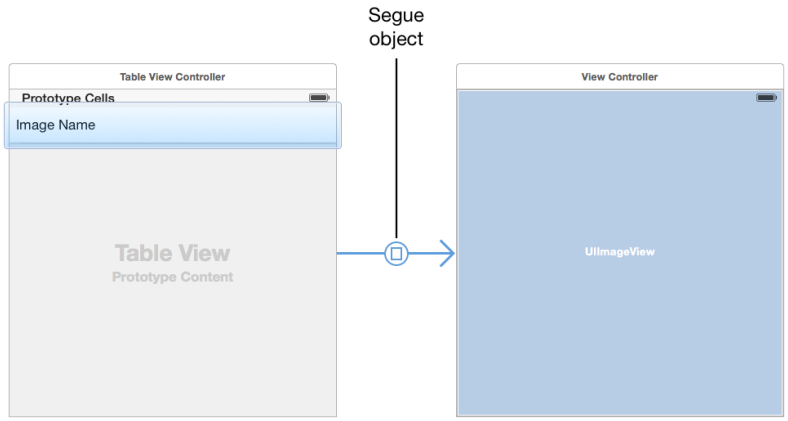
```
    @IBAction func touchDigit(_ sender: UIButton) {
        let digit = sender.currentTitle!
        if userIsInTheMiddleOfTyping {
            let textCurrentInDisplay = display.text!
            display.text = textCurrentInDisplay + digit
        } else {
            display.text = digit
        }
        userIsInTheMiddleOfTyping = true
    }

    @IBAction func performOperation(_ sender: UIButton) {
        if userIsInTheMiddleOfTyping {
            model.setOperand(operand: displayValue)
            userIsInTheMiddleOfTyping = false
        }
        if let methematicalSymbol = sender.currentTitle {
            model.performOperation(symbol: methematicalSymbol)
        }
        displayValue = model.result
    }
}
```

Multiple Views & View Controllers

- Segue
- Create a segue between View Controllers
- Table View Controller & its data source
- Use the prepare method to pass data between view controllers
- Embed a View Controller in a Navigation Controller

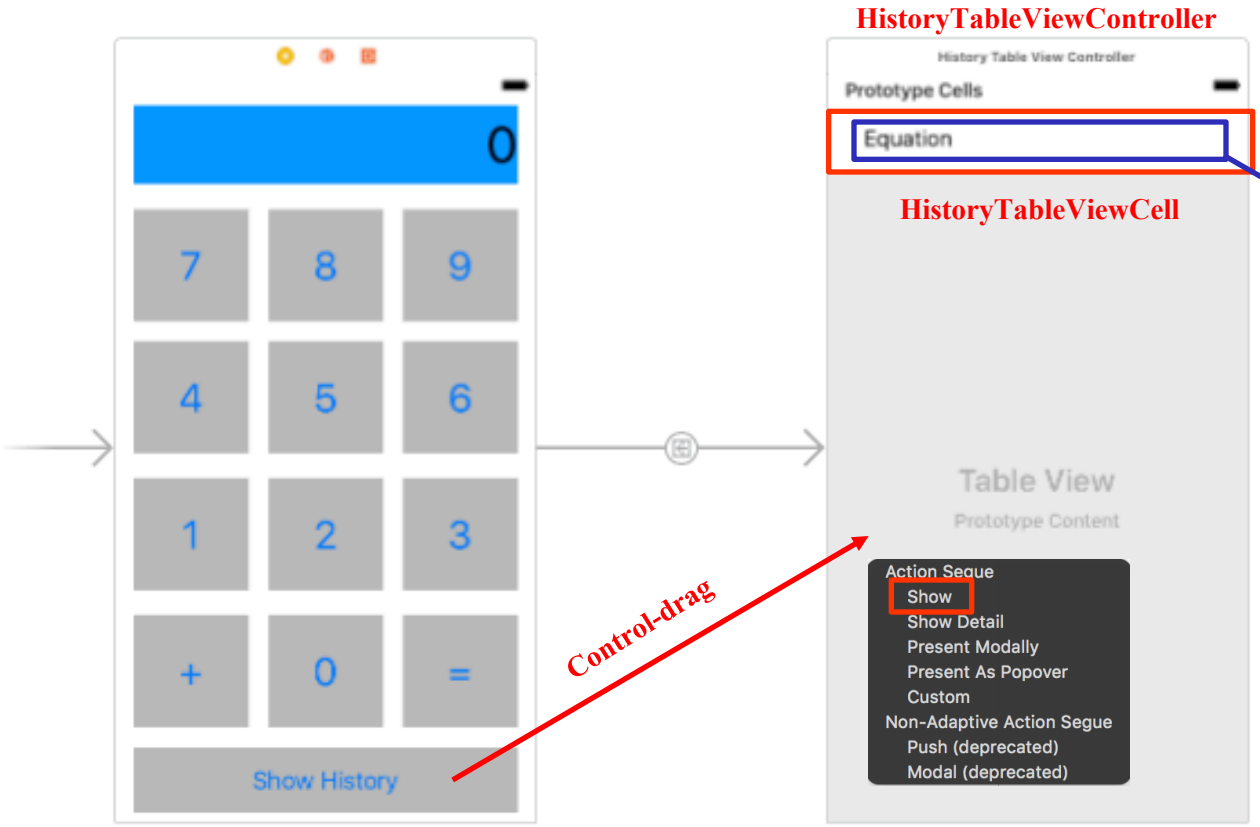
Segue

- A ***segue*** defines a transition between two view controllers in your app's storyboard file.
- The starting point of a segue is the button, table row, or gesture recognizer that initiates the segue.
- The end point of a segue is the view controller you want to display.



Create a segue between View Controllers

Let's say we want to add a "Show History" button at the bottom of the calculator view. And want to display each past equation as a separate row in a Table View.




```
class
HistoryTableViewCell:
UITableViewCell {

    @IBOutlet weak var
equationLabel: UILabel!

}
```

Table View Controller & its data source

```
var equations = [String]()

override func numberOfSections(in tableView: UITableView) -> Int {
        return 1 //return number of sections
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
    return equations.count //return number of rows
}

//To configure and set data for your cells
override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
"historyTableViewCell", for: indexPath)
    if let historyTableViewCell = cell as? HistoryTableViewCell {
        let equation = equations[indexPath.row]
        historyTableViewCell.equationLabel.text = equation
    }
    return cell
}
```

Question: Where does the *equations* data come from?
See next slide.

Pass data between view controllers

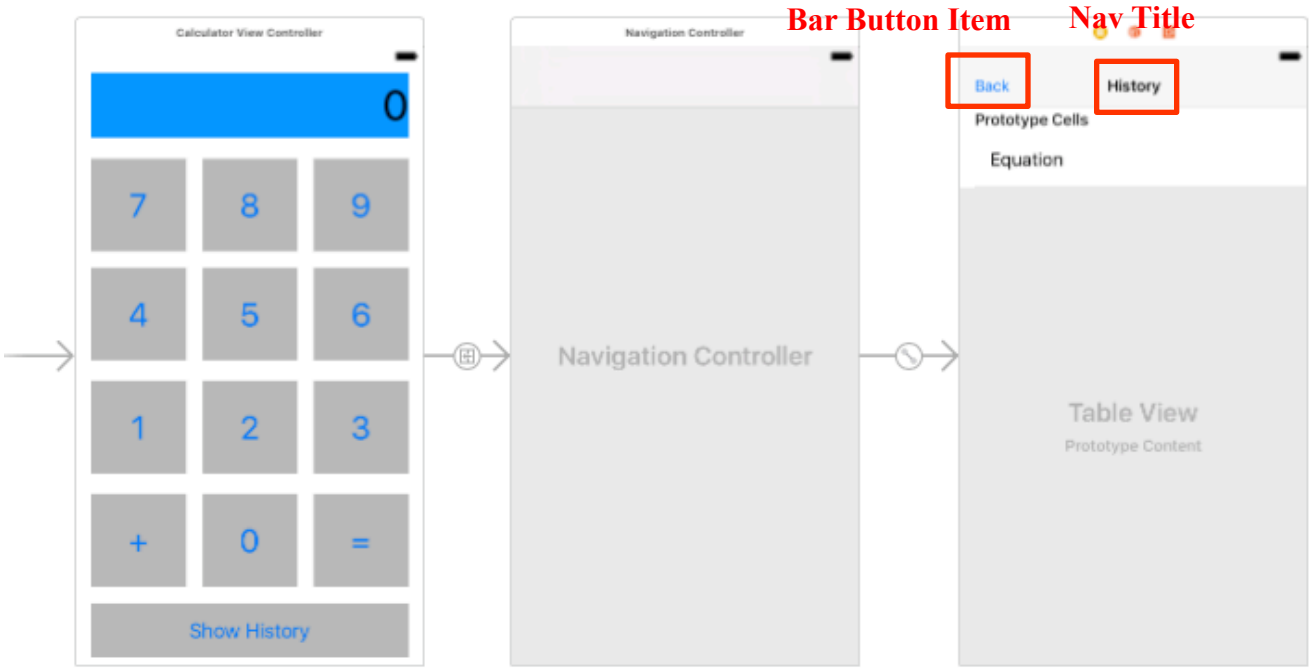
Add a ***prepare*** method in CalculatorViewController to "prepare for" the segue between CalculatorViewController and HistoryTableViewController.

```
// In a storyboard-based application, you will often want to do a
little preparation before navigation
 override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let historyTableViewController = segue.destination as?
HistoryTableViewController {
        historyTableViewController.equations = model.history
    }
 }
```

We also have to modify the model to save equations in history.

Embed a View Controller in a Navigation Controller

Navigation controller allows us to add a title and back button on the top of our history table.

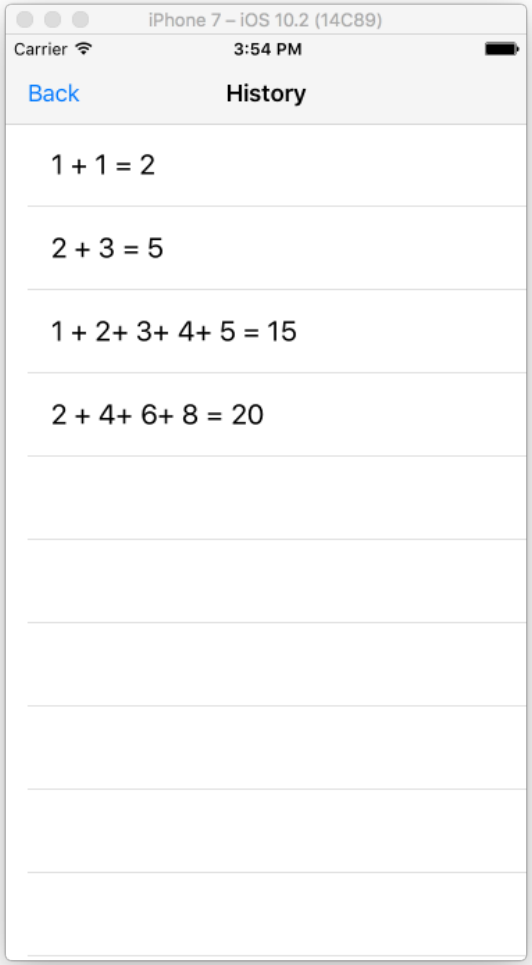


Select the Table View Controller, then choose Editor -> Embeded in -> Navigation Controller

You may find the Table View doesn't show history after this. Why?

Hint: Take a look at the "prepare" method. We need to update that method!

Demo



CocoaPods: Use External Dependencies

- CocoaPods introduction and install
- Add external dependencies
- Http networking and JSON parser

CocoaPods

- CocoaPods manages dependencies for your Xcode projects.
- You specify the dependencies for your project in a simple text file: your Podfile. CocoaPods recursively resolves dependencies between libraries, fetches source code for all dependencies, and creates and maintains an Xcode workspace to build your project.
- Install CocoaPods:

```
$ sudo gem install cocoapods
```

- To use it in your Xcode projects, run it in your project directory:

```
$ pod init
```

Add dependecies by CocoaPods

- Add dependencies in a text file named **Podfile** in your Xcode project directory

```
target ‘MyApp’ do
  use_frameworks!

  pod 'Alamofire'
  pod 'AlamofireSwiftyJSON'

end
```

- Install the dependencies in your project:

```
$ pod install
```

- Make sure to always open the Xcode workspace (*.xcworkspace) instead of the project file (*. xcodeproj) when you use CocoaPods with your project

Http networking and JSON parser

- We have installed Alamofire and AlamofireSwiftyJSON in the previous slide.
- Now we can use them for http networking and JSON parsing.

```
import Alamofire
import AlamofireSwiftyJSON
Alamofire.request("https://httpbin.org/get").responseSw
iftyJSON { response in
    let json = response.result.value //A JSON object
    let isSuccess = response.result.isSuccess
    if (isSuccess && (json != nil)) {
        //do something
    }
}
```

Http networking and JSON parser (cont'd)

- An example of use of the JSON parser

```
Alamofire.request("https://httpbin.org/get").responseSwiftyJSON {
response in
    let json = response.result.value //A JSON object
    let isSuccess = response.result.isSuccess
    if (isSuccess && (json != nil)) {
        let string = json!["stringData"].string //String?
        let number = json!["numberData"].intValue //Int
        let jsonObj = json!["objectData"] //JSON
        let array = json!["arrayData"].array //[JSON]?
        for element in array! {
            //each element is a JSON object
        }
        let innerElement = json!["1level"]["2level"] //JSON

    }
}
```

References

- A perfect IOS App example with step-by-step instructions
- IOS course by Stanford : Developing iOS 10 Apps with Swift
- iTunes U collections are moving to Podcasts
- The online Swift Language guide by Apple
- iBook: The Swift Programming Language (Swift 4)
- iBook: Using Swift with Cocoa and Objective-C (Swift 4)