
# iOS 13 Development Using Swift 5 and Xcode 11

www.techdata.com

# iOS/Swift Development

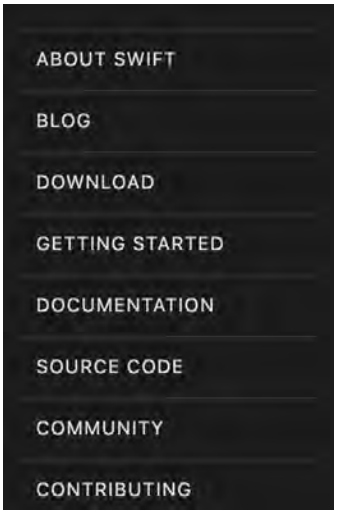
By Bear Cahill (BrainwashInc.com)



## Intro

| Version | Release Date | macOS | Linux |
|---|---|---|---|
| Swift 1.0 | September 9, 2014 | Yes | No |
| Swift 1.1 | October 22, 2014 | Yes | No |
| Swift 1.2 | April 8, 2015 | Yes | No |
| Swift 2.0 | September 21, 2015 | Yes | No |
| Swift 2.1 | October 20, 2015 | Yes | No |
| Swift 2.2 | March 21, 2016 | Yes | Yes |
| Swift 2.2.1 | May 3, 2016 | Yes | Yes |
| Swift 3.0 | September 13, 2016 | Yes | Yes |
| Swift 3.0.1 | October 28, 2016 | Yes | Yes |
| Swift 3.0.2 | December 13, 2016 | Yes | Yes |
| Swift 3.1 | March 27, 2017 | Yes | Yes |
| Swift 3.1.1 | April 21, 2017 | Yes | Yes |
| Swift 4.0 | September 19, 2017 | Yes | Yes |
| Swift 4.0.2 | November 1, 2017 | Yes | Yes |
| Swift 4.0.3 | December 5, 2017 | Yes | Yes |
| Swift 4.1 | March 29, 2018 | Yes | Yes |
| Swift 4.1.1 | May 4, 2018 | No | Yes |
| Swift 4.1.2 | May 31, 2018 | Yes | Yes |
| Swift 4.1.3 | July 27, 2018 | No | Yes |
| Swift 4.2 | September 17, 2018 | Yes | Yes |
| Swift 4.2.1 | October 30, 2018 | Yes | Yes |
| Swift 4.2.2 | February 4, 2019 | No | Yes |
| Swift 4.2.3 | February 28, 2019 | No | Yes |
| Swift 4.2.4 | March 29, 2019 | No | Yes |
| Swift 5.0[44] | March 25, 2019 | Yes | Yes |
| Swift 5.0.1 | April 18, 2019 | Yes | Yes |
| Swift 5.0.2 | July 15, 2019 | No | Yes |
| Swift 5.0.3 | August 30, 2019 | No | Yes |
| Swift 5.1 | September 10, 2019 | Yes | Yes |

Intro
Much more info at Swift.org




Intro
For iOS, we'll use Xcode from the App Store


Native vs Cross-Platform

Native

- Closer to the OS
- Access to system resources and Push Notifs
- Faster (in many cases)
- Less reuse
- Apple controlled
- Single-platform

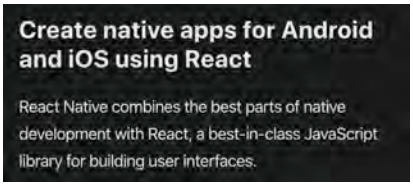


Native vs Cross-Platform

Cross-Platform/HTML

- Single code base
- Open to languages
- Develop once
- More reuse
- Potential to release without review
- Dependent upon 3rd party to update libraries

## Native vs Cross-Platform

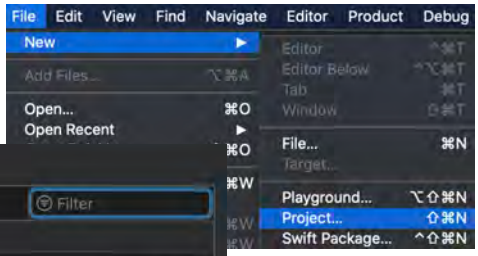



React Native

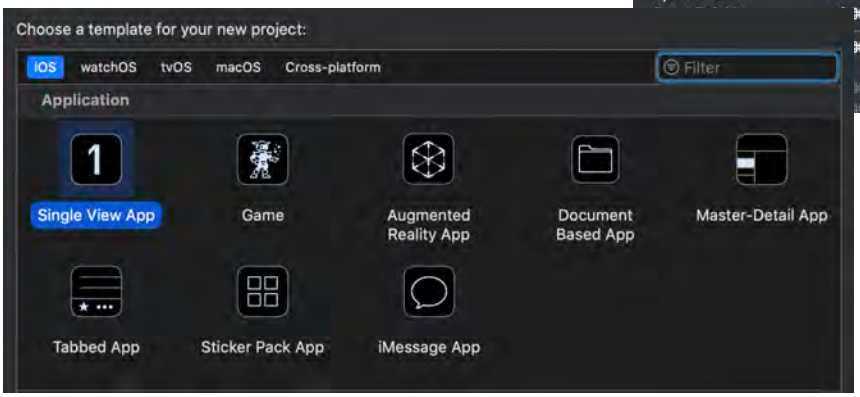
- See https://facebook.github.io/react-native/
- Uses native UI elements per platform
- Retains single code base
- JavaScript development



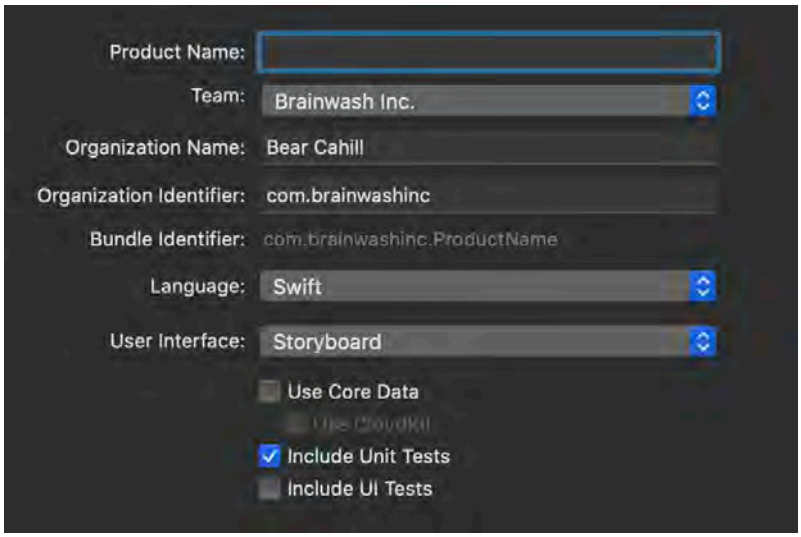
## Structure of an App
## Xcode > New Project


## Templates



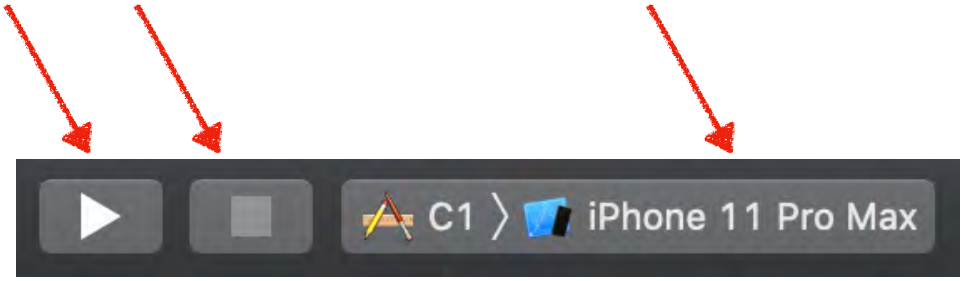
## Structure of an App
## Details of new Project






## Structure of an App
## Details of new Project

## Start/Stop project in selected Simulator



# Structure of an App
# View Controllers & Views

Inherit from UIViewController
UIViewController has property for view (UIView)

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the
            view.
    }


}
```

```
open class UIViewController : UIResponder,
    UIAppearanceContainer, UITraitEnvironm
    UIContentContainer, UIFocusEnvironment

    public init(nibName nibNameOrNil: Stri
        nibBundleOrNil: Bundle?)

    public init?(coder: NSCoder)

    open var view: UIView!
```



# Structure of an App
# Storyboard

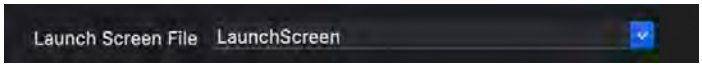
Views are visible, View Controllers have views

# Structure of an App Storyboard

Launch Storyboard is loading screen

Static

Also specified in Target settings

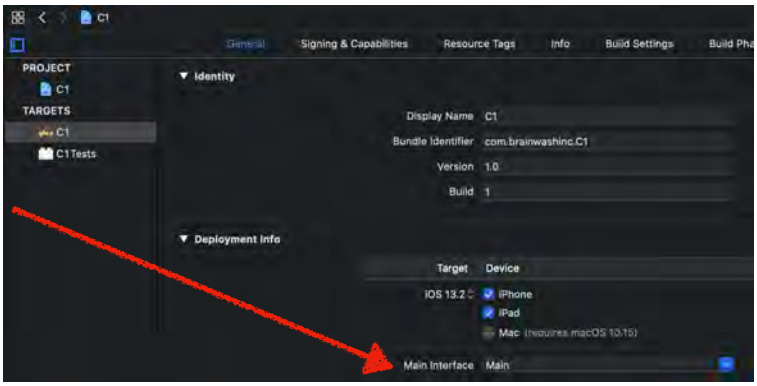




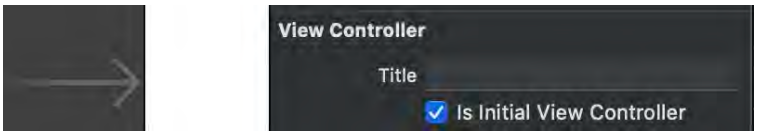


# Structure of an App Storyboard

Specified in General



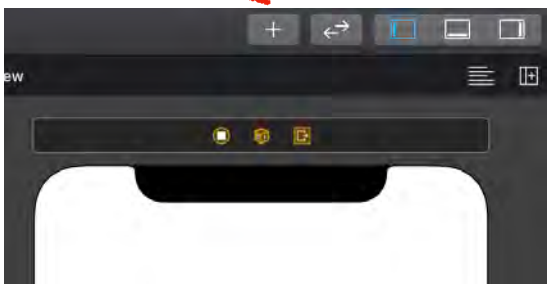

Storyboard specifies the Initial View Controller

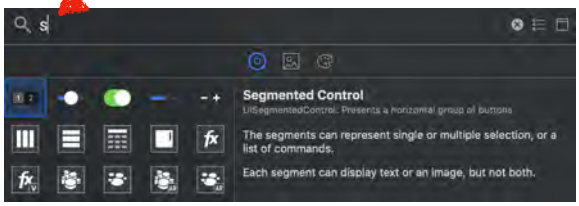


# Structure of an App
# Storyboard

UI items added visually via the Object Library Drag-n-Drop

Can be filtered:








# Structure of an App
# SwiftUI

Design UI in code (and visually) with live preview


Similarities to Storyboard/IB: Button, Nav Item, Text Field


Differences: List vs Table View, Text vs Lesson, etc.

# Structure of an App
# SwiftUI

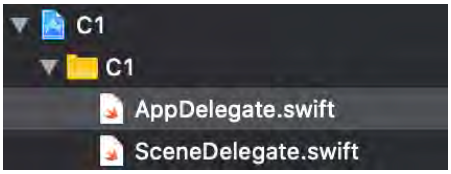
Layout with items like H/V/ZStack, Spacer

Change appearance with modifiers like font, padding

```
var body: some View {
    HStack {
        VStack {
            Text("Hello, World!")
            Text("Hello, World!2")
                .bold()
            Text("Hello, World!3")
        }
        Spacer()
        VStack {
            Text("Hello, World!11")
            Text("Hello, World!22")
                .fontWeight(.heavy)
            Text("Hello, World!33")
                .font(.footnote)
        }
    }
    .padding(60)
}
```




# Structure of an App
# App Delegate

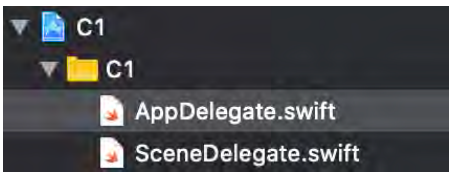



Generated for you with template

Handles system callbacks defined in UIApplicationDelegate

```
public protocol UIApplicationDelegate : NSObjectProtocol {

    @available(iOS 2.0, *)
    optional func applicationDidFinishLaunching(_ application:
        UIApplication)

    @available(iOS 6.0, *)
    optional func application(_ application: UIApplication,
        willFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool

    @available(iOS 3.0, *)
    optional func application(_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool
```

## Structure of an App
## Scene Delegate




Generated for you with template
Handles scene callbacks defined in UIWindowSceneDelegate

```
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?
```

Summary
Additional methods that you use to manage app-specific tasks occurring in a scene.

Declaration
`protocol UIWindowSceneDelegate`

Creates initial View and sets as window root view

```
// Create the SwiftUI view that provides the window contents.
let contentView = ContentView()

// Use a UIHostingController as window root view controller.
if let windowScene = scene as? UIWindowScene {
    let window = UIWindow(windowScene: windowScene)
    window.rootViewController = UIHostingController(rootView: contentView)
    self.window = window
    window.makeKeyAndVisible()
}
```



## Lab 1: Hello World


1. Create new project named HW
2. Open Main.storyboard
3. Open the Object Library
4. Drop a Label on your UI
5. Double click on the Label
6. Change text to “Hello World”
7. Run app in the Simulator

## Swift vs JavaScript

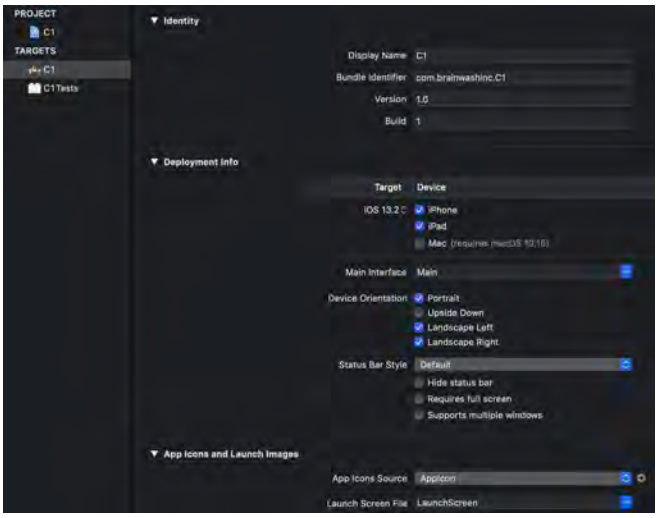
| | Swift | JavaScript |
|---|---|---|
| variables/ constants | var/let | var/const |
| Conditionals | No parens | Parens |
| try/catch | do-try-catch | try-catch |
| Various | No semi-colons, type inference, can't change type, conditionals don't evaluate, string interpolation | |



## Tour of Xcode
## Target Settings

Much is Defaulted/ Generated

Deployment Target
Orientation
Status Bar
Icon & Launch Screen




Frameworks, libraries, etc.

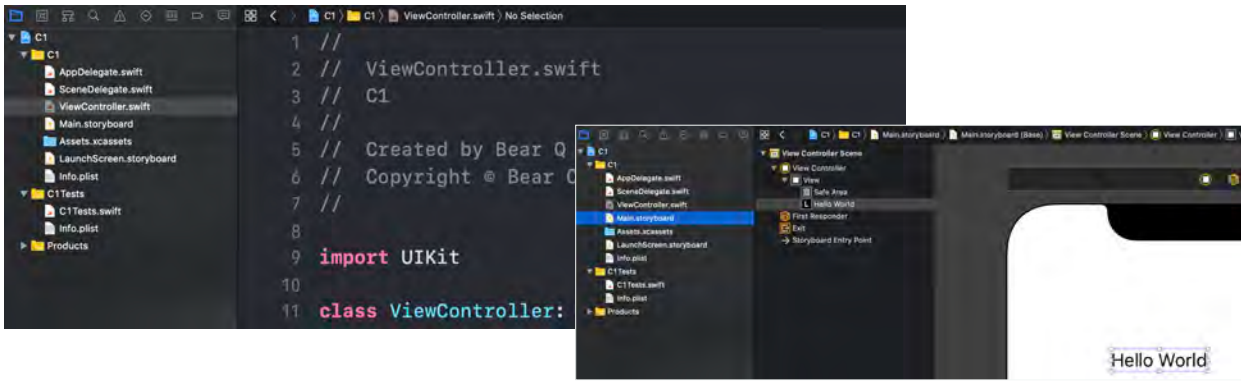
Tour of Xcode
Navigators

1. Project - project items (e.g., files)
2. Source Control - repo controls
3. Symbol - classes, methods, properties, etc.
4. Find - searches
5. Issues - build warnings/errors
6. Test - Unit Tests
7. Debug - runtime analytics
8. Breakpoint - breakpoints
9. Report - build reports

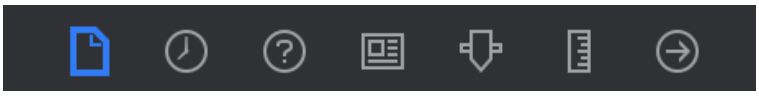


Tour of Xcode
Editor Area

Contents based on selection
Edit files, storyboards, db schema, etc.



Tour of Xcode
Inspectors

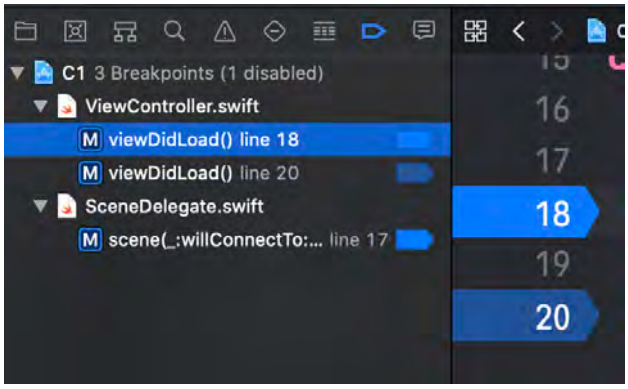


Contents based on selection

1. File - Localization, Target, etc.
2. History - repo history
3. Quick Help - help
4. Identity - class type
5. Attributes - attributes/properties
6. Size - layout
7. Connection - Outlets & Actions



Tour of Xcode
Debugging: Breakpoints



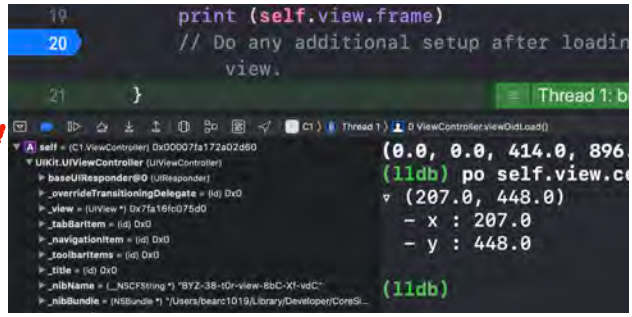

Set/enabled/disabled
in the left gutter

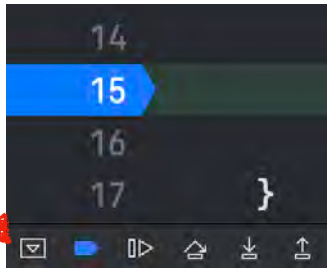
Can be managed and navigated to in Breakpoint Navigator

Tour of Xcode
Debugging: Debug Area

Shows values and output
Allows for commands
Debug Controls

1. Close
2. Deactivate
3. Continue
4. Step over
5. Step into
6. Step out






Lab 2: Debug

1. Open Lab 1(or solution provided Lab1.zip)
2. Add print line after super.viewDidLoad()
3. Add breakpoint on print line from step 2
4. Run app in Simulator
5. Step over print line
6. Verify printout
7. Enter command (e.g., po self.view)

Classes
Single Inheritance

Defined in any .swift file

Can override functions

Can call super functions

```
class ViewController: UIViewController {

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



Classes
Properties

```
class ViewController: UIViewController {

    let name = "Main"
    var isDirty : Bool = false
```

Type inferred

- Type can't change
- Type must be known at compile time

let - keyword for constant, good for memory mgmt

var - keyword for variable

Classes
Properties

Base types:

- Int - 32/64 bit, based on platform
- Double - 64 bit, Float is 32 bit
- String - Collection of Characters
- Bool - true/false



Classes
Initializers

Instances must be fully initialized

- Have default values or…
- Assigned values in initializer

Designated Initializer - creates fully initialized instance

Convenience Initializer - not designated, but calls one (eventually)

Classes
deinit

```
deinit {
    // write to database
}
```

Not mandatory
Doesn't clean up memory
Doesn't call super

Last chance before cleaned up in memory



Classes
Functions

```
func add(value1: Int, value2 : Int) -> Int {
    return value1 + value2
}
```

Starts with keyword "func"

Have…

- Name - "add"
- Parameters - "value1: Int"
- Return type - "-> Int"
  - If no return, "-> Void", "-> ()" or nothing

Classes
Functions

```
func add(this value1: Int, andThis value2 : Int) -> Int {
    return value1 + value2
}
```

Parameters have

- Argument Label - “this”, “andThis”
- Parameter Name - “value1”, “value2"

```
let result = add(this: 4, andThis: 8)
```

If one, it serves as both.
To not require parameter name when calling, use _

```
func add(_ value1: Int, _ value2 : Int) -> Int {
    return value1 + value2
}
```

```
let result = add(4, 8)
```



Classes
Functions

Parameters can have default values

```
func add(_ value1: Int, _ value2 : Int = 2) -> Int {
    return value1 + value2
}
```

Doesn’t require parameters with default values

```
let result = add
Int add(value1: Int)
Int add(value1: Int, value2: Int)
```

Functions and initializers are called similarly.

# Lab 3: Class/Function

1. In ViewController.swift, create a class named “TipCalc” (no base class/ inheritance)
2. Add one property (var) named tipPercentage (set the default value to 0.15)
3. Create a calcTip function with 1 parameter named “amount” of type Double and return type Double.
4. Create and store a TipCalc instance in ViewController’s viewDidLoad.
5. Call TipCalc’s calcTip function, store the result in a constant (let) & print it out.
6. Execute the app and verify the results.

```
class TipCalc {
    var tipPercentage = 0.15

    func calcTip(amount: Double) -> Double {
        return amount * tipPercentage
    }
}
```

```
override func viewDidLoad() {
    super.viewDidLoad()
    print (self.view.frame)
    let tc = TipCalc()
    let r = tc.calcTip(amount: 20.0)
    print (r)
```



# Lab 3: Class/Function - Extra

```
class TipCalc {
    var tipPercentage = 0.15

    init(tp : Double) {
        tipPercentage = tp
    }
```

1. Change the tipPercentage value.
    1. Run and test.
2. Create an initializer that takes a tipPercentage and sets the property.
    1. Remove the property from the class.
    2. Run and test.
3. Give the tipPercentage parameter a default value.
    1. Run and test with and without passing in a tipPercentage value.

```
class TipCalc {
    func calcTip(amount: Double, tipPercentage : Double = 0.15) -> Double {
        return amount * tipPercentage
    }
}
```

# Structs

Most base types (Int, Double, String) are structs

Similarities to Classes
- Properties
- Functions
- Initializers
- Can be extended (later)
- Can conform to protocols (later)

```
/// A double-precision, floating-point value type.
public struct Double {

    /// Creates a value initialized to zero.
    public init()
```

Differences
- No inheritance
- No deinit
- Passed by value (copied)
- No === (equality operator) because it's never the same instance



# More on Properties
# willSet/didSet Observers

```
class Person {
    var isDirty = false
    var DOB = Date()
    var age : Double = 0.0 {
        willSet {
            print (newValue)
            isDirty = true
        }
        didSet {
            DOB = Date.init(timeIntervalSinceNow: -age)
        }
    }
}
```

willSet/didSet Observers
- Called before/after property is set
- newValue/oldValue parameter

NOTE: DOB isn't updating age = infinite loop

More on Properties
Computed Properties

No stored value - doesn't take memory

Computed when called

Avoid unnecessary redundant values, sync

If only "get", no keyword needed

```
class Person {
    var isDirty = false
    var DOB : Date {
        return Date.init(timeIntervalSinceNow: -age)
    }
    var age : Double = 0.0 {
        willSet {
            print (newValue)
            isDirty = true
        }
    }
}
```



More on Properties
Computed Properties

Can have get and set

Get calculates it

Set stores in age

```
class Person {
    var isDirty = false
    var DOB : Date {
        get {
            return Date.init(timeIntervalSinceNow: -age)
        }
        set {
            age = -newValue.timeIntervalSinceNow
        }
    }
    var age : Double = 0.0 {
        willSet {
            print (newValue)
            isDirty = true
        }
    }
}
```

OOP in Swift

- Single Inheritance
- Protocols (POP)
- Similarities between Classes, Structs, and Enums
- Discussion



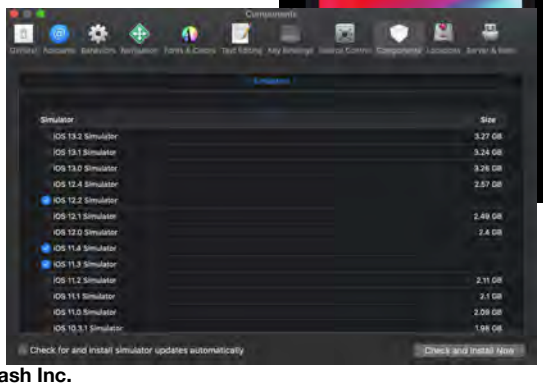
Simulator

Runs iOS - various versions and devices

Uses Mac CPU, RAM, disk, microphone, etc. (not camera)

App really run: Maps, Safari

Debug & Hardware menus are very helpful when testing



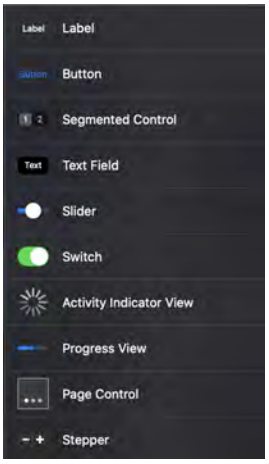
# UI Design
# Controls

Various UI controls for user input/display

Some have actions/events

Some have delegates

Some have both






# UI Design
# Controls

| Element | Property | Delegate | Event/Func |
|---|---|---|---|
| Button | | | Touch Up Inside |
| Segmented Control | Selected Segment Index | | Value Changed |
| Slider | Value | | Value Changed |
| Text Field | Text | UITextFieldDelegate | Should Return, Editing Changed |

## UI Design
## Controls

| Element | Property | Delegate | Event/Func |
|---|---|---|---|
| Swift | isOn | | Value Changed |
| Picker View | | UIPickerView Datasource/Delegate | Num components, rows, title, did… |
| Stepper | Value | | Value Changed |
| | | | |



## UI Design
## Views

Anything visual inherits from UIView and can set those attributes

Size and position defined in Frame property

Can have subviews (property)

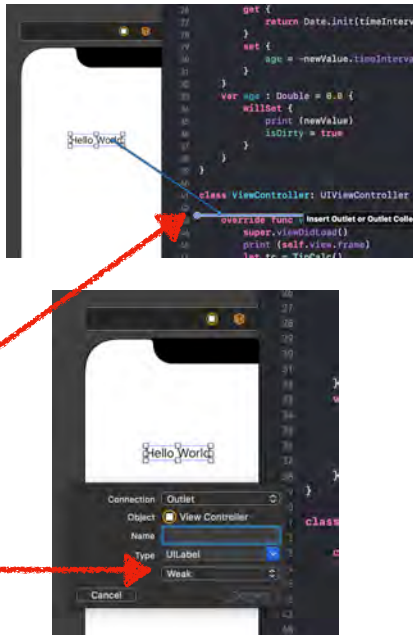
Can receive touches and add Gestures

# UI Design
# Outlets

UI elements can be created in code
If created visually, Outlets connect UI to code for access.

Editor > Assistant - side-by-side

Control+drag - UI into code
Weak reference to voice retain cycle






# Lab 4: Outlet

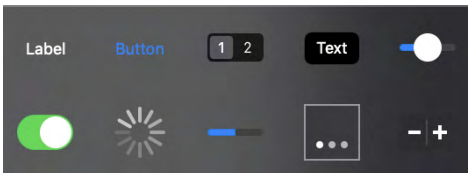
1. Open Lab3 project (or Lab3.zip solution)
2. Open Main.storyboard
3. Open Assistant (Editor > Assistant) - verify ViewController.swift is on right
4. Control+drag “Hello World” label into ViewController class (see previous slide)
5. Match the settings in the image including the name of lblInfo
6. Verify the outlet is created as a property
7. Change the text property on the lblInfo reference
8. Run the app and verify the text is displayed.



```
class ViewController: UIViewController {

    @IBOutlet weak var lblInfo: UILabel!
    override func viewDidLoad() {
        super.viewDidLoad()

        lblInfo.text = "Hello iOS!"
```

UI Design
Actions

Some elements have events




Buttons - e.g., Touch Up Inside (once)
Slider - e.g., Value Changed (continuous)

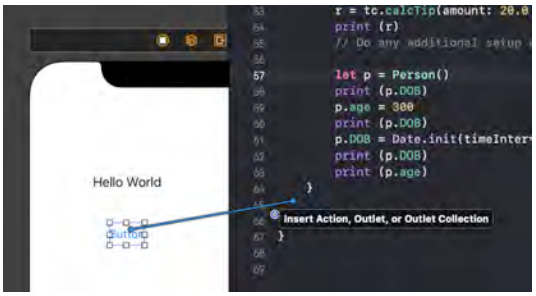
Actions can call functions on events

Element is passed into function



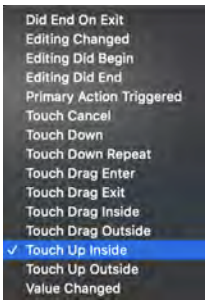
UI Design
Actions

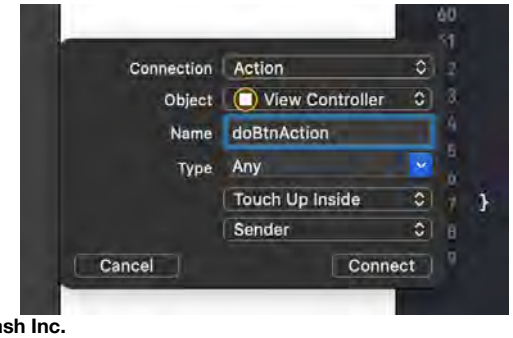
Control+drag element into code




Create Action
Name and Parameter Type

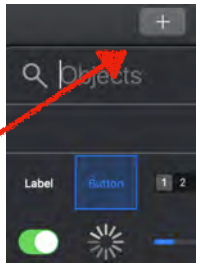
Event



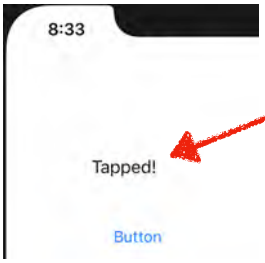



# Lab 5: Action

1. Open Lab4 project (or Lab4.zip solution)
2. Open Main.storyboard
3. Open Assistant (Editor > Assistant) - verify ViewController.swift is on right
4. Add a Button to your UI from the Object Library
5. Control+drag button into ViewController class (see previous slide)
6. Create Action function in code to change lblInfo.text
7. Run the app and verify the text is displayed.




```
@IBAction func doBtnAction(_ sender: Any) {
    lblInfo.text = "Tapped!"
}
```






# UI Design
# TextField

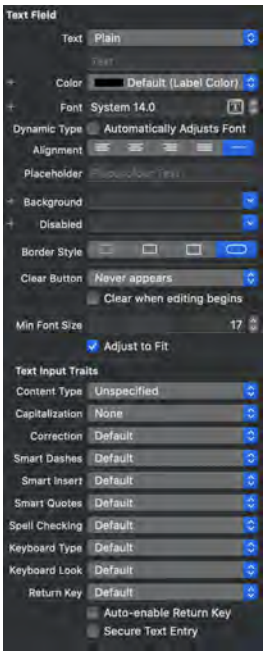
# Text input from user: become/ resignFirstResponder

# Text Input Trait Attributes

# Delegate for actions
# UITextFieldDelegate

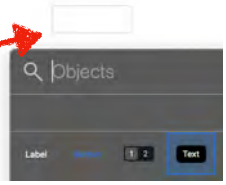
# Assign and Implement

```
public protocol UITextFieldDelegate : NSObjectProtocol {

    @available(iOS 2.0, *)
    optional func textFieldShouldBeginEditing(_ textField

    @available(iOS 2.0, *)
    optional func textFieldDidBeginEditing(_ textField: U

    @available(iOS 2.0, *)
    optional func textFieldShouldEndEditing(_ textField:
        responder status. NO to disallow the editing sess

    @available(iOS 2.0, *)
    optional func textFieldDidEndEditing(_ textField: UIT
        removed from window) or endEditing:YES called

    @available(iOS 10.0, *)
    optional func textFieldDidEndEditing(_ textField: UIT
        of textFieldDidEndEditing:
```



# Lab 6: TextField

1. Open Lab5 project (or Lab5.zip solution)
2. Open Main.storyboard
3. Add a TextField to the UI
4. Create an Outlet in the code (named tfInput)
5. Update doBtnAction to set the lblInfo.text property to the value in tfInput.text
6. Run the app and verify the text is displayed.




```
@IBOutlet weak var tfInput: UITextField!
```

```
@IBAction func doBtnAction(_ sender: Any) {
    lblInfo.text = tfInput.text
}
```




# Lab 6: TextField - Extra

1. Set the Delegate outlet on the TextField to be the ViewController
2. Declare to implement UITextFieldDelegate in the ViewController
3. Implement the shouldReturn function from the protocol to return false
4. Also, set the lblInfo.text to the tfInput.text
5. Run the app and verify the text is displayed.


```
class ViewController: UIViewController, UITextFieldDelegate {
```


```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    lblInfo.text = tfInput.text
    return false
}
```

## Lab 7: Tip Calculator

1. Design (paper, etc.) a Tip Calculator considering the various controls possible to allow the user to best input the values needed and see output.
2. Open Lab6 project (or Lab6.zip solution)
3. Open Main.storyboard
4. Design the UI including various outlets and actions
5. Run the app and test the UI and functionality.



## UI Discussion

- Compare to CSS
- CSS-in-JS
- Android
- Other

Collections
Array

Ordered
Non-Unique

```
var names : [String] = ["Bear", "Schmeb"]
names.append("Jedidiah")
names.remove(at: 0)
print (names[0])
```

Declared: Array<Type> or [<Type>]
Empty array created with above and ()

Accessed via subscript: names[4]

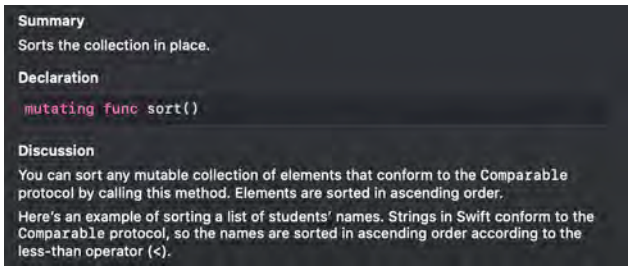
let non-mutable; var mutable



Collections
Array

Functions:
append - element or collection
remove - various
count - number of elements
first/last
+ - must be 2 arrays

In-place vs. Return
sort(ed) - Comparable
reverse(d)



Collections
Set

Unique, Unordered - sort func returns Array
Declared: Set<Type>, create with ()

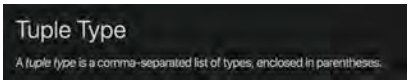
No subscript, Common funcs: insert, count, contains

Set functions:

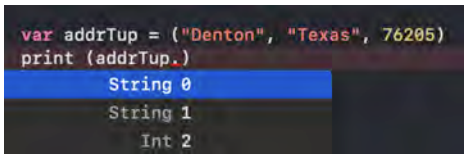
- intersection, formIntersection
- isSubset, isStrictSubset
- isSuperset
- subtracting
- union



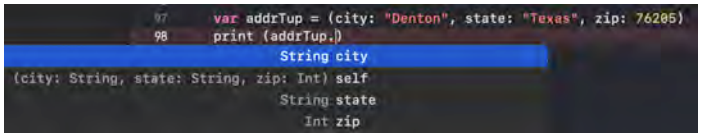
Collections
Tuples




Multiple values grouped together




Accessed via order index or name




Can be decomposed to variables

```
let (city, state, zip) = addrTup
```

Collections
Dictionary

Key, Value pairs

Declared: Dictionary<Type:Type>  or <Type, Type>
Key type is Hashable: String, number, etc.

Accessed/Set via subscript with Key Type

```
var locations = [76205: "Dentn", 70448: "Mandeville"]
locations[76205] = "Denton"
print (locations[76205])
```



Collections
Dictionary

```
locations.removeValue(forKey: 76201)
locations[76201] = nil
```

Remove with func or set to nil

Some funcs return tuples:
  first, remove, etc.

```
let firstZip = locations.first.k
                          Int key
```

Iterating uses tuples

Can access keys and values with .keys and .values

Collections
String

Collection of Characters - iterable
+ - must be two Strings

Characters accessed via String.Index

String Interpolation - within a String literal, anything in \() is evaluated and added to the string

```
print ("I live in \(city) and that has \(city.count) characters.")
```

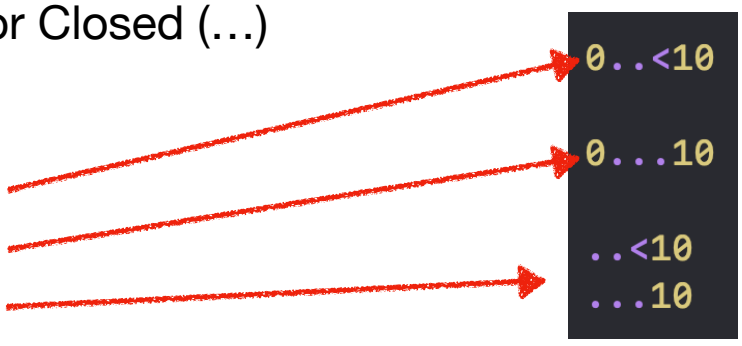


Control Flow

Ranges:
- Lower-bound
- Upper-bound
- Inclusive or Closed (…)

```
let r = 0..<10
```

Partial:
- Up-to
- Through
- From



Loops

```
var names : [String] = ["Bear", "Schmeb"]
```

for-in - collection or range
for subscript

```
for name in names {
    print (name)
}
```

```
for x in 0..<names.count {
    print (names[x])
}
```

while - condition

```
let someTime : TimeInterval = 23423
while Date().timeIntervalSince1970 < someTime {
    print ("Waiting...")
}
```

repeat-while - condition
checked AFTER iterations

```
repeat {
    print ("Waiting...")
} while Date().timeIntervalSince1970 < someTime
```



Switch

Can check any type including tuples

Must be exhaustive

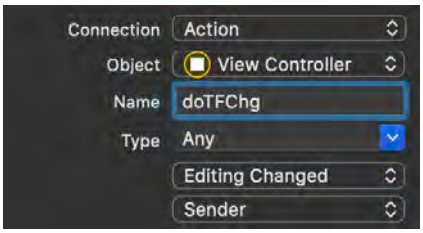
No break

fallthrough forces (only) next condition

```
switch addrTup {
case ("Denton", "Texas", 76205):
    print ("Near UNT")
case (_, "Texas", let zip):
    print ("Somewhere in Texas")
    print (zip)
    fallthrough
case (_, _, 70000), (_, _, 80000):
    print ("70000 or 80000")
default:
    print ("Other")
}
```

## Lab 8: Tip Calculator V2 - Collections

1. Use a Collection to store the percentage options
2. Use a segmented control to display percentage options (loop) and action to calculate.
3. Create action from Text Field into code. Implement the value changed function and calculate tip when changed.
4. Experiment and test!






## UI Design: Auto-Layout Constraints

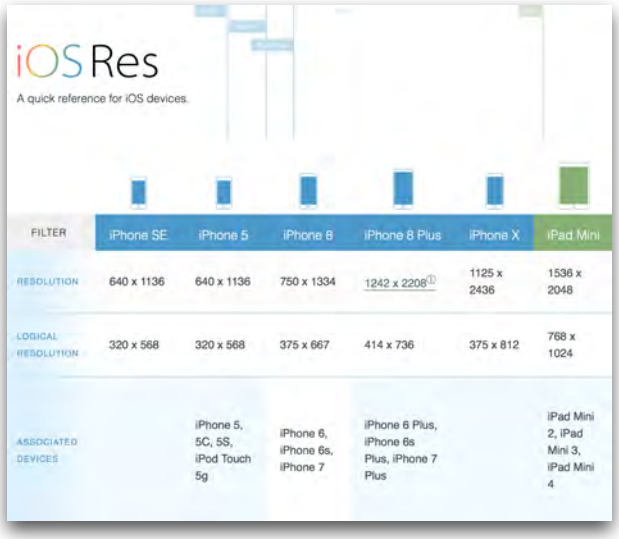
## Many Devices

| Device | Native Resolution (Pixels) | UIKit Size (Points) |
|---|---|---|
| iPhone X | 1125 x 2436 | 375 x 812 |
| iPhone 8 Plus | 1080 x 1920 | 414 x 736 |
| iPhone 8 | 750 x 1334 | 375 x 667 |
| iPhone 7 Plus | 1080 x 1920 | 414 x 736 |
| iPhone 6s Plus | 1080 x 1920 | 375 x 667 |
| iPhone 6 Plus | 1080 x 1920 | 375 x 667 |
| iPhone 7 | 750 x 1334 | 375 x 667 |
| iPhone 6s | 750 x 1334 | 375 x 667 |
| iPhone 6 | 750 x 1334 | 375 x 667 |
| iPhone SE | 640 x 1136 | 320 x 568 |
| iPad Pro 12.9-inch (2nd generation) | 2048 x 2732 | 1024 x 1366 |
| iPad Pro 10.5-inch | 2224 x 1668 | 1112 x 834 |
| iPad Pro (12.9-inch) | 2048 x 2732 | 1024 x 1366 |
| iPad Pro (9.7-inch) | 1536 x 2048 | 768 x 1024 |
| iPad Air 2 | 1536 x 2048 | 768 x 1024 |
| iPad Mini 4 | 1536 x 2048 | 768 x 1024 |

- From https://developer.apple.com/library/content/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/

# UI Design: Auto-Layout Constraints

Many Screens




- From http://iosres.com/Displays/Displays.html

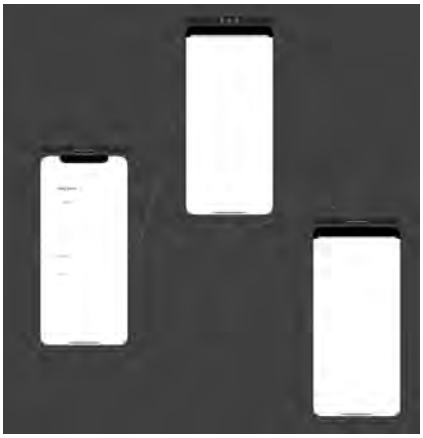


# UI Design: Auto-Layout Constraints

Storyboards - Main and Launch, stored as XML

Storyboards have Scenes
A Scene have 1 View Controller
One “Initial View Controller” (attribute)

Constraints setup rules to instruct the system about UI



UI Design: Auto-Layout
Constraints

Segues - Connect Scenes/View Controllers

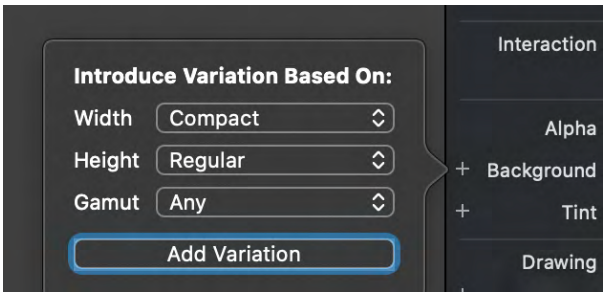
Segues are “Adaptive” to the device/form factor they’re running on (e.g., Landscape, Portrait, iPad)

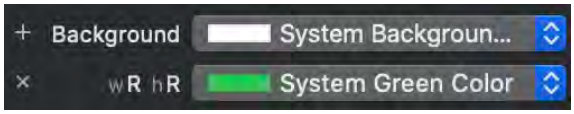
Defined in Compact and Regular height and width



UI Design: Auto-Layout
Variants

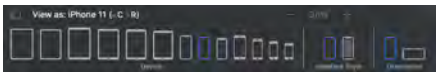
Variants for Height, Width and Gamut



## UI Design: Auto-Layout Constraints

Design can be based and changed for various device size/orientation



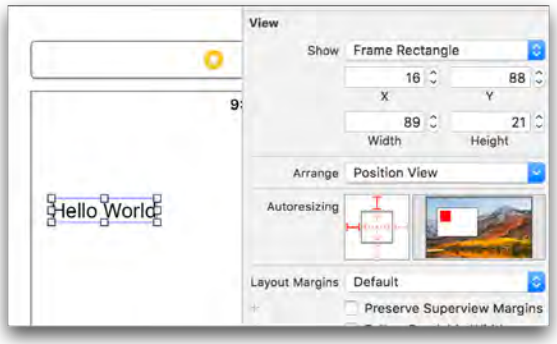
Can be Previewed
(Editor > Preview)



## UI Design: Auto-Layout Constraints

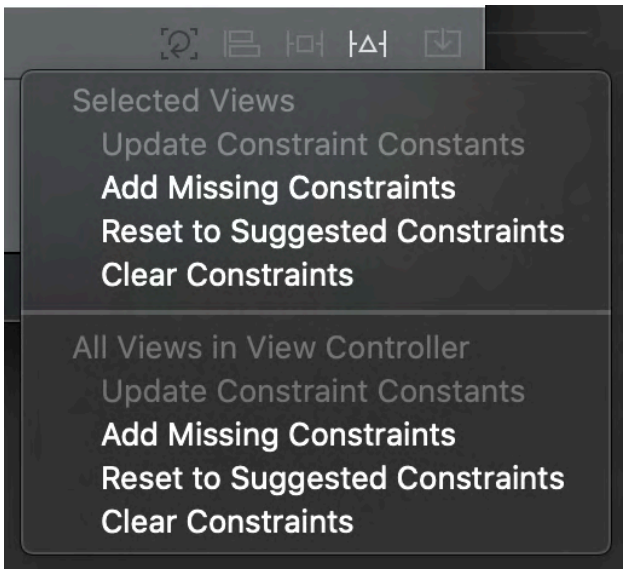
Auto-Resizing mask

Converted to Constraints



# UI Design: Auto-Layout Constraints

Resolve Auto-Layout Issues






# UI Design: Auto-Layout Constraints

Add New Constraints

# UI Design: Auto-Layout Constraints

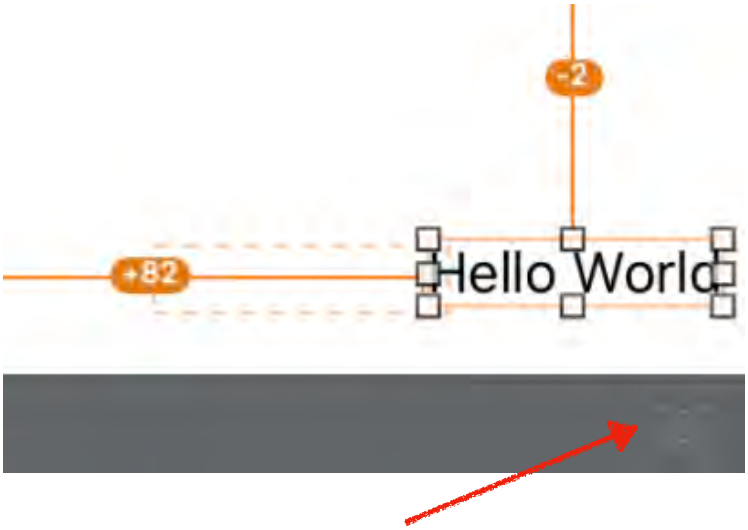
Add New Alignment Constraints




# UI Design: Auto-Layout Constraints

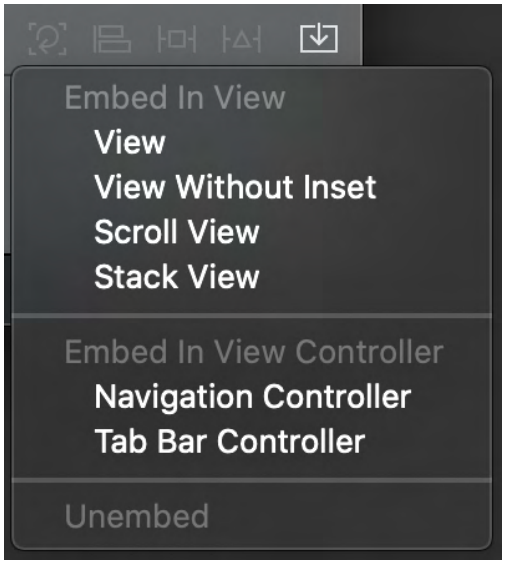
Update Frames

- Misplaced frames



# UI Design: Auto-Layout Constraints

Embed In View



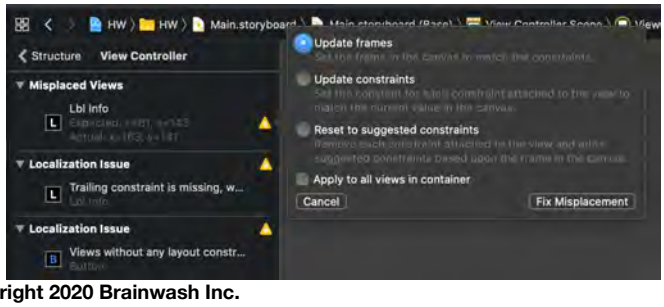



# UI Design: Auto-Layout Constraints

Can create constraints by control dragging from one UI element to another

Issues indicated with yellow or red circle

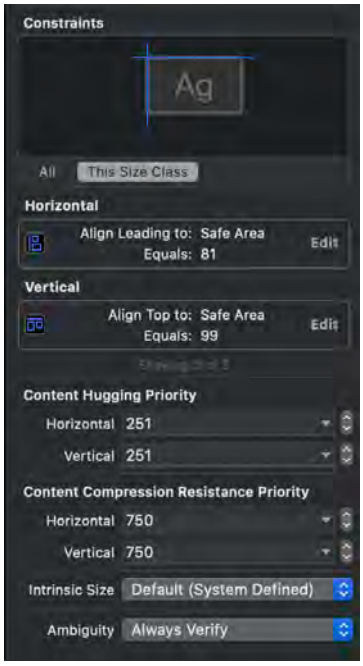
Warning listed often with possible solutions

# UI Design: Auto-Layout Constraints

Fixes may include

- More constraints
- Fewer constraints
- Changes to existing constraints
- Constraint, hugging or compression resistance priorities






# UI Design: Auto-Layout Stack Views

Group Views together to be managed as arranged subviews

## Lab 9: Tip Calculator V3 - Constraints

1. Update your Tip Calculator UI with constraints
2. Design for all devices/orientations

- Consider keyboard type and placement
- Consider variants for screen sizes




## Constraints in Code

Relative to other views' anchors and guides

Anchors:
leadingAnchor/trailingAnchor
leftAnchor/rightAnchor
topAnchor/bottomAnchor
widthAnchor/heightAnchor
centerXAnchor/centerYAnchor
firstBaselineAnchor/lastBaselineAnchor

Constraints need to be activated (can be deactivated)

# Constraints in Code

Constraints can be created with NSLayoutConstraint init

```
// set the leading edge to be 44 points in from the leading edge of the self.view safe area
// created and activated
NSLayoutConstraint.init(item: lblInfo!, attribute: .leading, relatedBy: .equal, toItem:
    self.view.safeAreaLayoutGuide, attribute: .leading, multiplier: 1.0, constant: 44)
    .isActive = true
```

…and activated

From one item's attribute to another with relation

| Parameters | |
|---|---|
| view1 | The view for the left side of the constraint. |
| attr1 | The attribute of the view for the left side of the constraint. |
| relation | The relationship between the left side of the constraint and the right side of the constraint. |
| view2 | The view for the right side of the constraint. |
| attr2 | The attribute of the view for the right side of the constraint. |
| multiplier | The constant multiplied with the attribute on the right side of the constraint as part of getting the modified attribute. |
| c | The constant added to the multiplied attribute value on the right side of the constraint to yield the final modified attribute. |



# Constraints in Code

```
// set the top of the labe to be 43 points below the safe area of the self.vew
let lblConTop = lblInfo.topAnchor.constraint(equalTo: self.view.safeAreaLayoutGuide.topAnchor, constant: 43)
lblConTop.isActive = true
```

Constraints can be created from the anchors

Still need to be activated

```
let tfCons = [
    // 28 below bottom of label
    tfInput.topAnchor.constraint(equalTo: lblInfo.bottomAnchor, constant: 28),
    tfInput.leadingAnchor.constraint(equalTo: self.view.safeAreaLayoutGuide.leadingAnchor, constant: 44),
    tfInput.trailingAnchor.constraint(equalTo: self.view.safeAreaLayoutGuide.trailingAnchor, constant: -44)
]
NSLayoutConstraint.activate(tfCons)
```

Can be activated in bulk (array)

## Constraints in Code

Constraints can have priorities

Helps when handling conflicts

```
lblConTrailing.priority = UILayoutPriority.init(500)
lblConTrailing.priority = .
    UILayoutPriority defaultHigh
    UILayoutPriority defaultLow
    UILayoutPriority dragThatCannotResizeScene
    UILayoutPriority dragThatCanResizeScene
    UILayoutPriority fittingSizeLevel
    UILayoutPriority required
    UILayoutPriority sceneSizeStayPut
    UILayoutPriority init(rawValue: Float)
```

There are constants or you can use Floats

Some use contacts

```
btnCalc.widthAnchor.constraint(equalToConstant: 100),
```

… or relate (relative with multiplier) to other values.

```
// half the width of segTipPercs
btnCalc.widthAnchor.constraint(equalTo: segTipPercs.widthAnchor, multiplier: 0.5)
```



## Constraints in Code

layoutMarginsGuide:
Respect the insets of the parent view


So if the insets change the UI will be relative to that:

```
self.view.directionalLayoutMargins = NSDirectionalEdgeInsets(top: self.view.directionalLayoutMargins.top,
                                                             leading: 10,
                                                             bottom: self.view.directionalLayoutMargins.bottom,
                                                             trailing: 100)

// text field
let tfCons = [
    // 28 below bottom of label
    tfInput.topAnchor.constraint(equalTo: lblInfo.bottomAnchor, constant: 28),
    tfInput.leadingAnchor.constraint(equalTo: self.view.layoutMarginsGuide.leadingAnchor, constant: 44),
    tfInput.trailingAnchor.constraint(equalTo: self.view.layoutMarginsGuide.trailingAnchor, constant: -44)
]
```

## Constraints in Code

readableContentGuide:

This layout guide defines an area that can easily be read without forcing users to move their head to track the lines.

safeAreaLayoutGuide:

When the view is visible onscreen, this guide reflects the portion of the view that is not covered by navigation bars, tab bars, toolbars, and other ancestor views.



## Lab 10: Tip Calculator V4 - Constraints in Code

1. Update your Tip Calculator UI with constraints created in code

## SwiftUI

Initial ContentView is created for you

```
import SwiftUI

struct ContentView: View {
```

And injected in SceneDelegate

```
// Create the SwiftUI view that provides the window contents.
let contentView = ContentView()

// Use a UIHostingController as window root view controller.
if let windowScene = scene as? UIWindowScene {
    let window = UIWindow(windowScene: windowScene)
    window.rootViewController = UIHostingController(rootView: contentView)
    self.window = window
    window.makeKeyAndVisible()
}
```

It conforms to View (protocol)

Views are a function of “state”

State changes result in
UI changes

```
public protocol View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required `body` property.
    associatedtype Body : View

    /// Declares the content and behavior of this view.
    var body: Self.Body { get }
}
```



## SwiftUI

Can add items from library or directly in code

Items have modifiers: .font, .padding, .color, etc.
Including size, scale and position modifiers

Visual preview can be interactive with “live preview” (only works in Portrait)

Preview managed in code

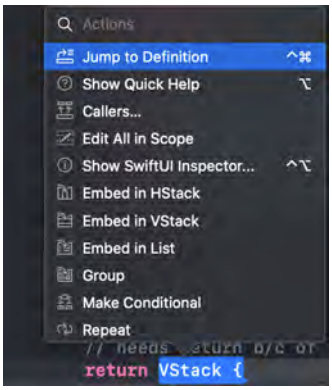
```
struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

## SwiftUI

Items can be organized in stacks: HStack, VStack, ZStack

Inits for those have parameters alignment and spacing

Context menu (⌘+click) allows for actions like embedding in other stacks






## SwiftUI

List is like table view

Text is like Label

TextField is like TextField

Button is like Button

```
var body: some View {
    VStack {

        List{
            Section(
                header:ListHeaderView(orderModel:self.orderModel,text:"Music")
            ){
                ForEach(orderModel.orders){item in
                    OrderRowView(orderItem:item)
                }
                .onDelete(perform:delete)
            }
        }
    }
}
```

But… some are different:
Picker is like Picker and Segment Control (and others) based on style

Create custom “View” items and include in your view/stacks

## SwiftUI

@State

```
@State private var billString = "0.00"
@State private var billAmt = 0.00
```

```
TextField("Enter Bill Amount", text: $billString)
```



## SwiftUI

@ObservedObject

Similar to @State but not local to one view
Usually for more complex types (e.g., has properties)

@Published

Counter part to @ObservedObject
Declares it's published to the @ObservedObject versions

## SwiftUI

```
struct MenuDetailView: View {

    @EnvironmentObject var settings:UserPreferences
```

@EnvironmentVariable

Available to a view and any subviews without passing

Where a View is created, value is passed in

@EnvironmentVariable property needs the value (or crash)

```
let contentView = RootTabView().environmentObject(UserPreferences())
```



## SwiftUI

Visually many, many things are different: creating, configuring, managing, etc.

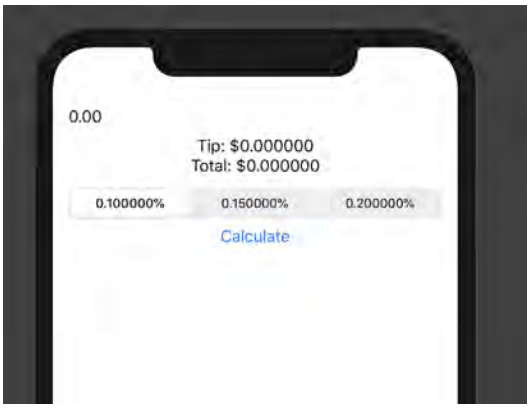
Data-wise things are very, very different: binding, state, observed, etc.

Approach is different also: consider state driving UI, events driving logic/state

Lab 11: Tip Calculator V5 - SwiftUI

1. Refactor your Tip Calculator using SwiftUI

- See Lab11.zip for help






Thinking Swifty
Error/Throw

Declared as throws
Anything that implements Error (empty) protocol

```swift
func mightFail(x: Int) throws {

    if x < 0 {
        throw NSError(domain: "My Error",
                      code: 42,
                      userInfo: ["something":543])
    }

}
```

```swift
enum MyError : Error {
    case bad, old, small
}

func check() throws {
    throw MyError.bad
}
```

Thinking Swifty
do/try/catch

Anything that throws should be called with try
Or the Error/exception should be propagated

Can check for specific types

```
func tryToFail() {
    do {
        try mightFail(x: -5)
    }
    catch {
        print (error.localizedDescription)
    }
}
```

```
func tryToFail() {
    do {
        try mightFail(x: -5)
    }
    catch let err as MyError {
        print (err)
    }
    catch {
        print (error.localizedDescription)
    }
}
```



Thinking Swifty
do/try/catch

Funcs that throw can be treated as Optional returns

```
func retIntOrFail(str : String) throws -> Int {
    if let i = Int(str) {
        return i
    }
    throw MyError.bad
}

func tryToFail() {
    let x = try? retIntOrFail(str: "junk")
    print ("\(x)") ⚠ String interpolation produces a debug
}
```

```
func tryToFail() {
    let x = (try? retIntOrFail(str: "junk")) ?? 0
    print ("\(x)")
}
```

Thinking Swifty
Enumerations

```
enum Direction {
    case up, down, left, right
}
```

enum keyword
No default underlying type

```
let dir : Direction = .
return_am
    Direction down
    Direction left
    Direction right
    Direction up
```

Scoping helps with auto-complete

Switches (exhaustive) can cover all cases

Prints the value: “up” (not 0)



Thinking Swifty
Enumerations

```
enum Direction {
    case up(Int)
    case down(Int)
    case left(String)
    case right(String, Int)
}
```

```
let d = Direction.right("West", 5)
```

Associated Values

```
func testDir(d: Direction) {
    switch d {
    case .up(let val):
        print ("up \(val)")
    case .down(_):
        print ("down")
    case .left(let v1):
        print (v1)
    case .right(let str, let rv):
        print ("\(str) and \(rv)")
    @unknown default:  ⚠ Default will never be executed
        print ("Unknown... for now!")
    }
}
```

## Thinking Swifty Enumerations

Can have Base type

```
let d = Direction.ini
    M Direction? init(rawValue: Int)
fun Creates a new instance with the specified raw
}   value.
```

Provides init with rawValue (may return nil)

Can access raw value

```
print (d?.rawValue)
```



## Thinking Swifty Enumerations

Enums can have functions, implement protocols, computed properties

Great for exclusive values (e.g., error codes)

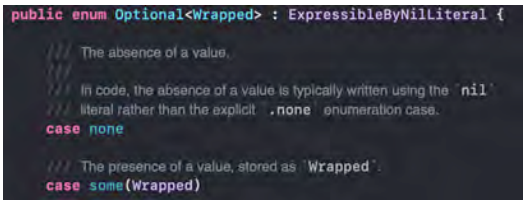
```
enum Direction : Int, CaseIterable {
    case up
    case down
    case left
    case right

    var max : Int { Direction.allCases.count - 1 }

    func printRaw(d: Direction) {
        print ("\(d.rawValue)")
    }
}
```

Thinking Swifty
Optionals

Values can’t be nil in Swift… Optionals can wrap nil

Enum with two cases:

- .none (nil)
- .some(Wrapped) - Wrapped is a Generic






Thinking Swifty
Optionals

Declaration:

| | |
|---|---|
| Optional<Type> | var name : Optional<String> |
| Type? | var name : String? |
| Type! | var name : String! // Implicit! |

Thinking Swifty
Optionals

Unwrapping:

Optional Binding

```
// unwraps and assigns: !nil is true
if let n = name {
    print (n)
}
```

Optional Chaining

```
// stops if any nil is hit
if let val = name?.reversed().first?.asciiValue?.byteSwapped {
    print (val)
}
```



Thinking Swifty
Optionals

Unwrapping:

Nil-Coalescing
Operator

```
// use name if not nil, otherwise, rhs
let finalName = name ?? "no name"
```

Unconditional (Danger)

```
// crashes if name is nil
let len = name!.count
```

Thinking Swifty
Optionals

Properties can be Optionals

Func parameters can be Optionals

Func/init can return Optionals
  Accessing values thru first, last, etc. - may not exist

Good with Guard statement (coming soon)



Thinking Swifty
Operators & Types

Operators generally deal with defined types
  + is for 2 Ints, Doubles or Strings
  Context sensitive

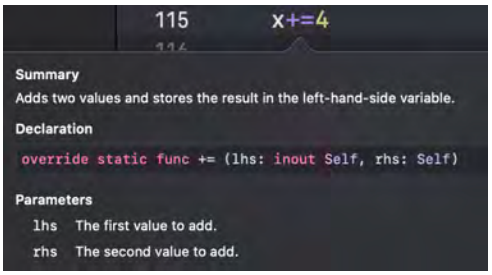

Operators are functions

```
/// - Parameters:
///   - lhs: The first value to add.
///   - rhs: The second value to add.
public static func + (lhs: Int, rhs: Int) -> Int
```

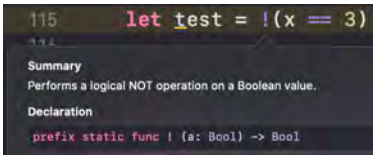
Thinking Swifty
Operators & Types

```
/// - Parameters:
///   - lhs: The first value to add.
///   - rhs: The second value to add.
public static func + (lhs: Int, rhs: Int) -> Int
```

Infix - between params
has lhs and rhs
+, -, *, %




Postfix - between params
has lhs and rhs, saves in lhs




Prefix - comes before



Thinking Swifty
Operators & Types

Unary - one target: -, !

Binary - two targets: +, -, *, /, =, %

Ternary - three targets: ?

Thinking Swifty
Operators & Types

&&, ||, !

Assignment: =

Compound Assignment: *=, +=, etc.

Comparison: >, <, !=, == (equal),
=== (identity, classes only)



Thinking Swifty
Guard

```
guard x == 3 else { return }
```

Only the true shall pass…
  Otherwise, return or be thrown!

If false, must return control: return or
  throw exception

May have multiple lines

```
guard x == 3 else {
    print ("failed: returning")
    return
}
```

## Thinking Swifty
## Guard

Optional binding in a Guard stays in scope

```
func readServer(url : String?) {
    guard let urlString = url else { return }

    guard let theURL = URL(string: urlString) else { return }
```

Can combine multiple checks/binding

```
func readServer(url : String?) {
    guard let urlString = url,
        urlString.count > 5,
        let theURL = URL(string: urlString) else { return }
```



## Thinking Swifty
## Guard

Guards are a great way to communicate the definitive need of a given case to be true.

If/else can work but doesn't communicate.

And if the return statement is deleted, compile error!

## Thinking Swifty
## Defer

Code to be performed before returning control to the caller.

Stack - LIFO

Create for cleanup

```
func write(data : Data, toPath path : String) -> String? {
    guard let fileHandle = FileHandle.init(forUpdatingAtPath: path)
       else { return "Error opening file" }

    // now that the file is open, set up defer
    // performed second (or only if exception in write)
    defer {
        fileHandle.closeFile()
    }

    fileHandle.write(data)

    // performed first
    defer {
        print ("write success!")
    }

    return nil // success
}
```



## Thinking Swifty
## Access Control

- Based on modules (build target) and source files
- open/public - useable within a module or when imported
- internal - only within module (default)
- fileprivate - only within source file
- private - only within declaration (or extensions in same file)
- @testable - special import for unit tests

## Thinking Swifty
## Access Control

> Open access is the highest (least restrictive) access level and private access is the lowest (most restrictive) access level.
>
> Open access applies only to classes and class members, and it differs from public access by allowing code outside the module to subclass and override, as discussed below in Subclassing. Marking a class as open explicitly indicates that you've considered the impact of code from other modules using that class as a superclass, and that you've designed your class's code accordingly.

From : https://docs.swift.org/swift-book/LanguageGuide/AccessControl.html



## Thinking Swifty
## Typealias

Declare a type as an alias to another type

```
typealias Name = String
typealias Address = [String:String]
typealias Age = Double
```

Good for functions and closures too!

## Lab 12: Tip Calculator V6 - Enum/Op/Guard

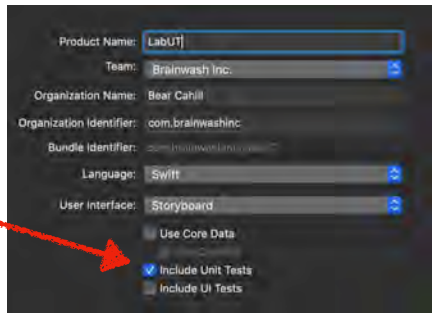
1. In any project or playground…
2. Create an Enum named NumError that declares to implement Error with 3 cases: tooSmall, tooBig, other
3. Create an Enum named NumCode with an underlying type of Int and 4 cases: zero, one, two, three
4. Write a func that takes one parameter of type Int, throws and returns a NumCode
5. Guard: If the Int parameter is not less than 4, throw NumError.tooBig
6. Convert the Int parameter to a NumCode.
7. If it results in a nil, throw an applicable NumError.

- Check Lab11.zip for solution/help.

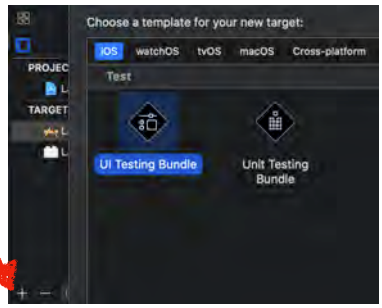


## Unit Testing
## Unit Tests

Added when creating a project



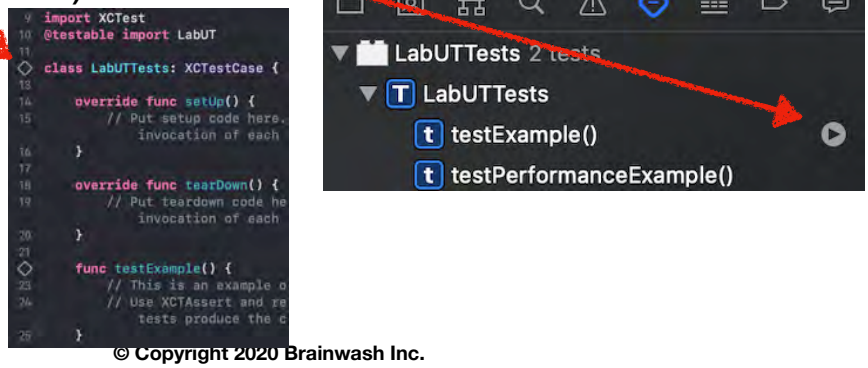

And/Or later as a Target Bundle



Unit Testing
Unit Tests

Tests can be run from the Test Navigator

Or in the file (gutter)



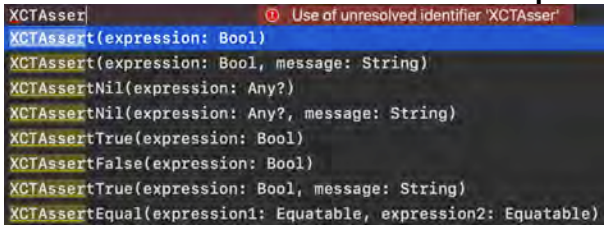



Unit Testing
Unit Tests

setUp and tearDown are called for each test

Any func starting with “test” will be considered a test

XCTAssert funcs are used to determine pass/fail



Unit Testing
Performance Tests

Anything with “testPerformance” is a performance test

self.measure will be run 10 times and timed

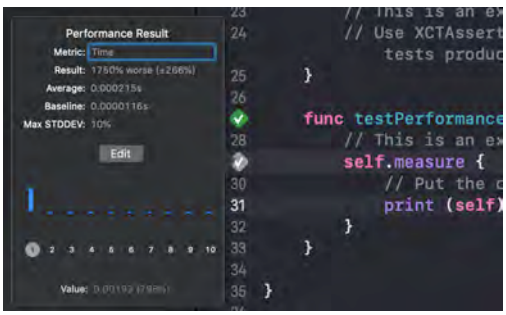
Then you have a baseline

```
func testPerformanceExample() {
    // This is an example of a performance test case.
    self.measure {
        // Put the code you want to measure the time of here.
    }
}
```



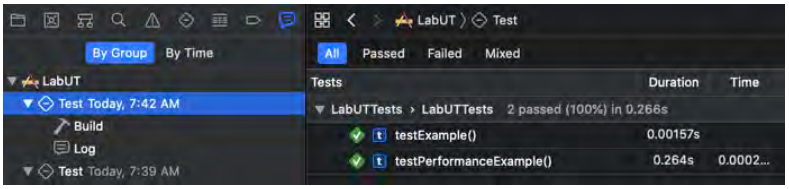

Unit Testing
Performance Tests

Edit the baseline and max std deviation for future runs




Unit Testing
Performance Tests

Results can be seen in the Report Navigator






Unit Testing
UI Tests

Created at Project creation and/or added as Target Bundle

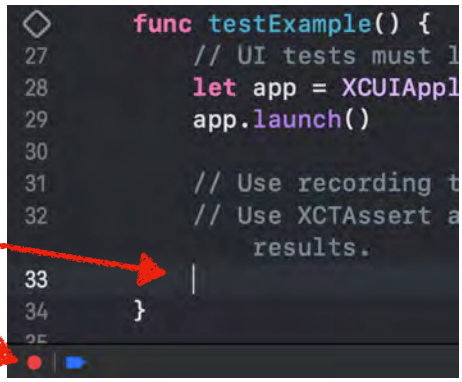
Launches app for each test


```
func testExample() {
    // UI tests must launch the application that they test.
    let app = XCUIApplication()
    app.launch()

    // Use recording to get started writing UI tests.
    // Use XCTAssert and related functions to verify your tests produce the correct
        results.
}
```

Unit Testing
UI Tests

Place cursor
and click record button

Code for UI actions is
generated as you use app






Unit Testing
UI Tests

Not always good code

Not the object types you're use to

```
app.buttons["Button"].tap()
app.buttons["Second"].tap()

app.sliders["50%"].swipeLeft()
app.buttons["Second"].tap()
app.buttons["Button"].tap()
```

```
func testExample() {
    // UI tests must launch the appl
    let app = XCUIApplication()
    app.launch()

    // Use recording to get started
    // Use XCTAssert and related fu
        results.

    let app = XCUIApplication()
    app.buttons["Button"].tap()
    app.buttons["Second"].tap()

    let app = XCUIApplication()
    app.sliders["50%"].swipeLeft()
    app.buttons["Second"].tap()
    app.buttons["Button"].tap()
}
```

## Lab 13: Unit Tests

1. Create a new project and add Unit Tests
2. Change “testExample” to another name starting with “test”
3. Create a value stored in a variable.
4. Use various XCTAssert calls to test for pass/fail.

- See Lab13.zip for example.

```
func testOne() {
    let dbl = Double("1") ?? 0
    XCTAssertEqual(dbl, 1, "should be 1")
}
```



## Patterns
## Extensions

Add to existing code (class, struct, enum or protocol):

- Functionality
- Computed properties
- Initializers
- Subscripts
- Protocol conformance

```
extension String {
    func encrypted() -> String {
        return String(self.reversed())
    }
}
```

Patterns
Extensions

Examples:
- UIImage - scale itself
- UIColor - add “backgroundColor” function
- UIViewController - add “displayLoading” UI
- String - add “displayAlert” function
- Protocol (next) conformance

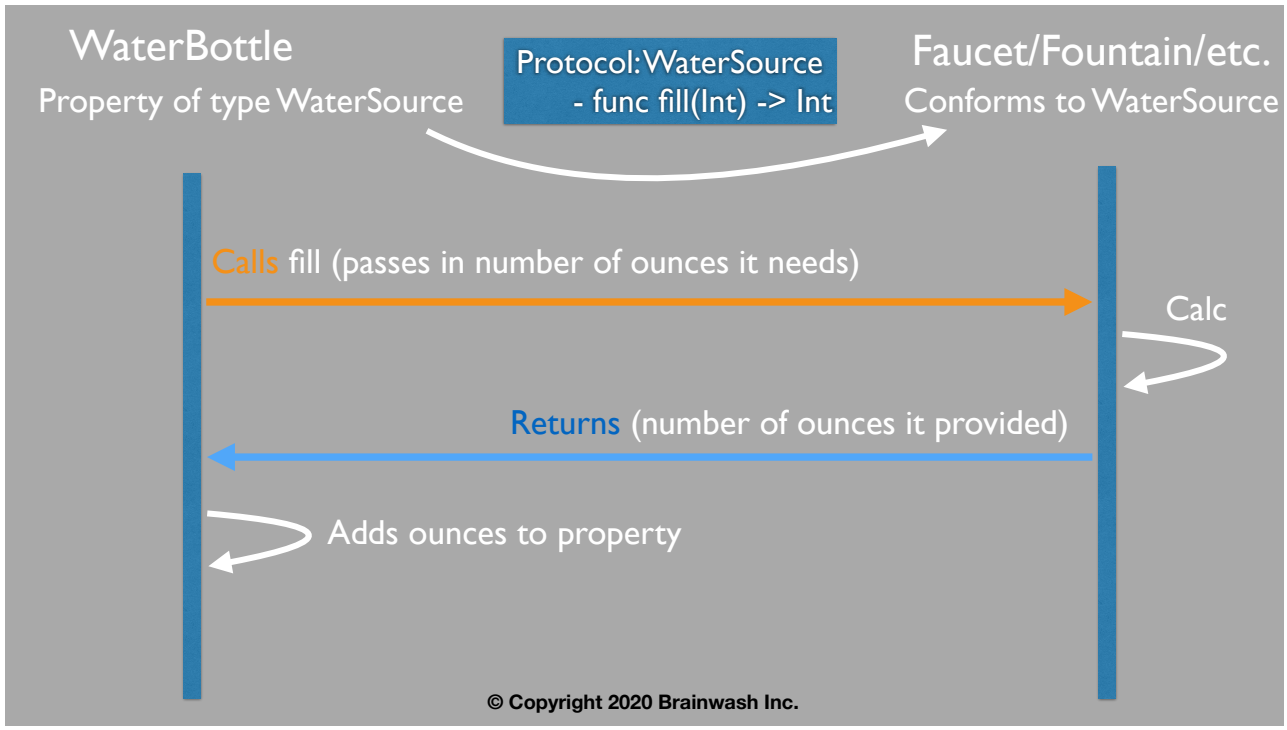


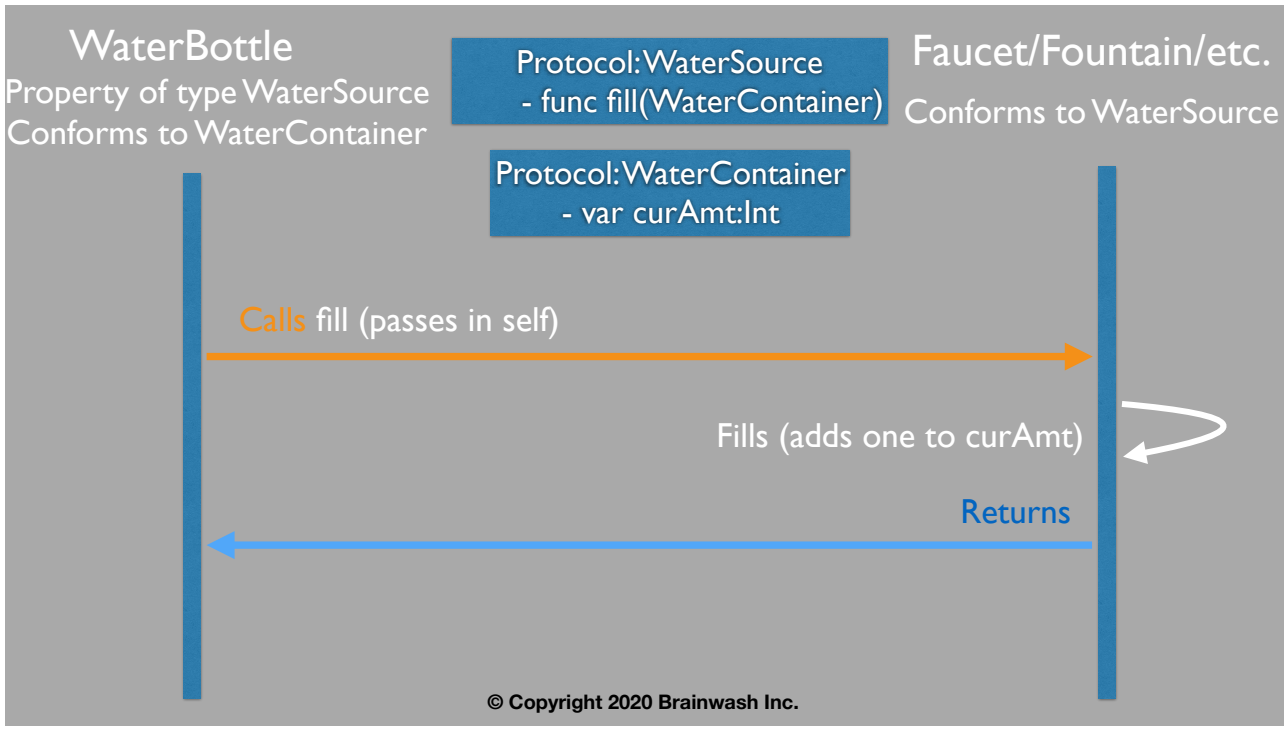
Patterns
Protocols

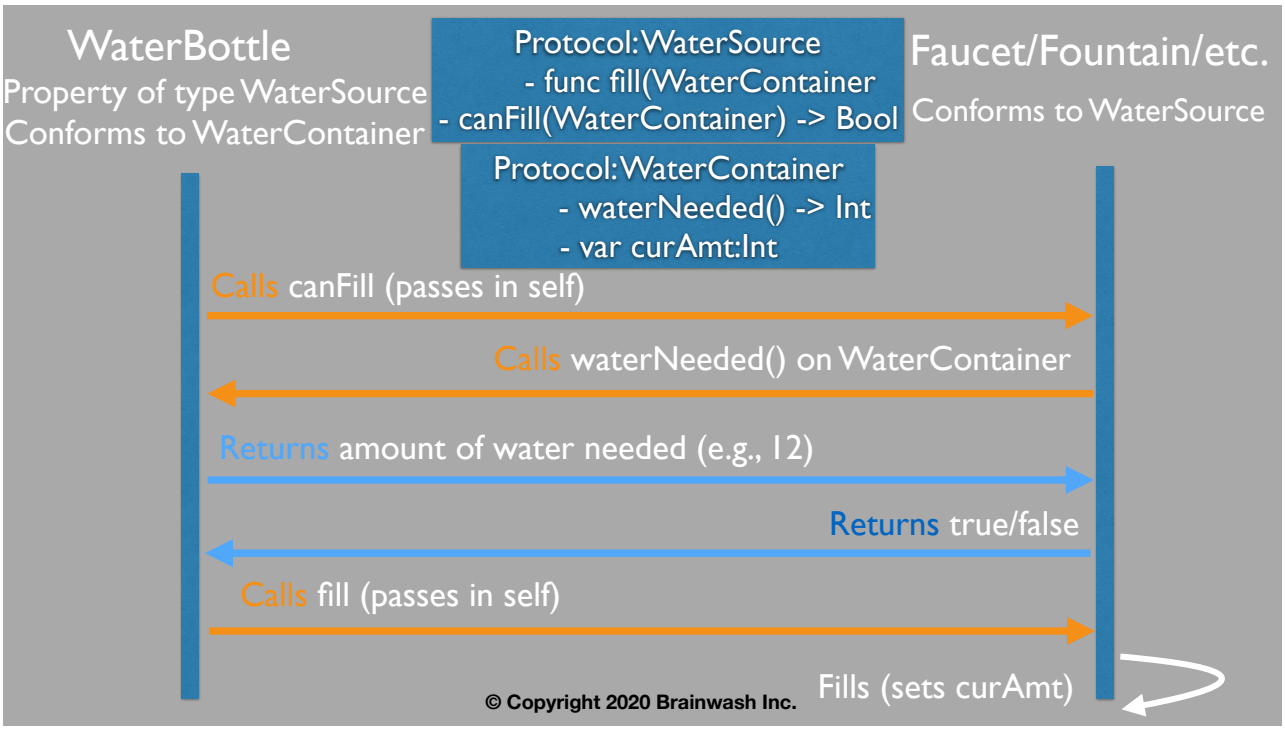
Defines a interface/blueprint of functions/properties
- Types the adopt protocol, must implement it

Can be extended, inherit and have default implementations (in Extension)

Types that implement a Protocol can be passed around as *that* Protocol-Type - Polymorphism


WaterBottle
Property of type WaterSource
Protocol:WaterSource
- func fill(Int) -> Int
Faucet/Fountain/etc.
Conforms to WaterSource
Calls fill (passes in number of ounces it needs)
Calc
Returns (number of ounces it provided)
Adds ounces to property
© Copyright 2020 Brainwash Inc.


WaterBottle
Property of type WaterSource
Conforms to WaterContainer
Protocol:WaterSource
- func fill(WaterContainer)
Faucet/Fountain/etc.
Conforms to WaterSource
Protocol:WaterContainer
- var curAmt:Int
Calls fill (passes in self)
Fills (adds one to curAmt)
Returns
© Copyright 2020 Brainwash Inc.




Patterns
Delegate

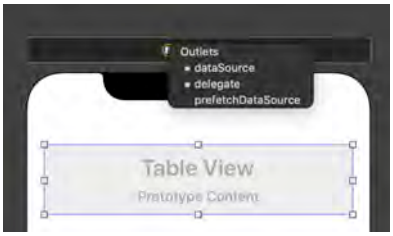
Think of the Water Bottle example but the Water Bottle (or similar) class has a property for the Water Source.

Classes like Table View, Picker View, Map View, Location Manager, Web View, UIApplication, Text Field, etc. all have a “delegate” property to call for actions/data/etc. when necessary. That delegate’s type is a protocol for your class to implement and be set.

Table View
Table

Handles UI - scrolling, recycling cells, swipes, etc.

Needs information/data/action: height for row, number of sections/rows, cell for row, didSelect,…




Properties for UITableViewDelegate, UITableViewDatasource

Static tables in Table View Controller don't change.

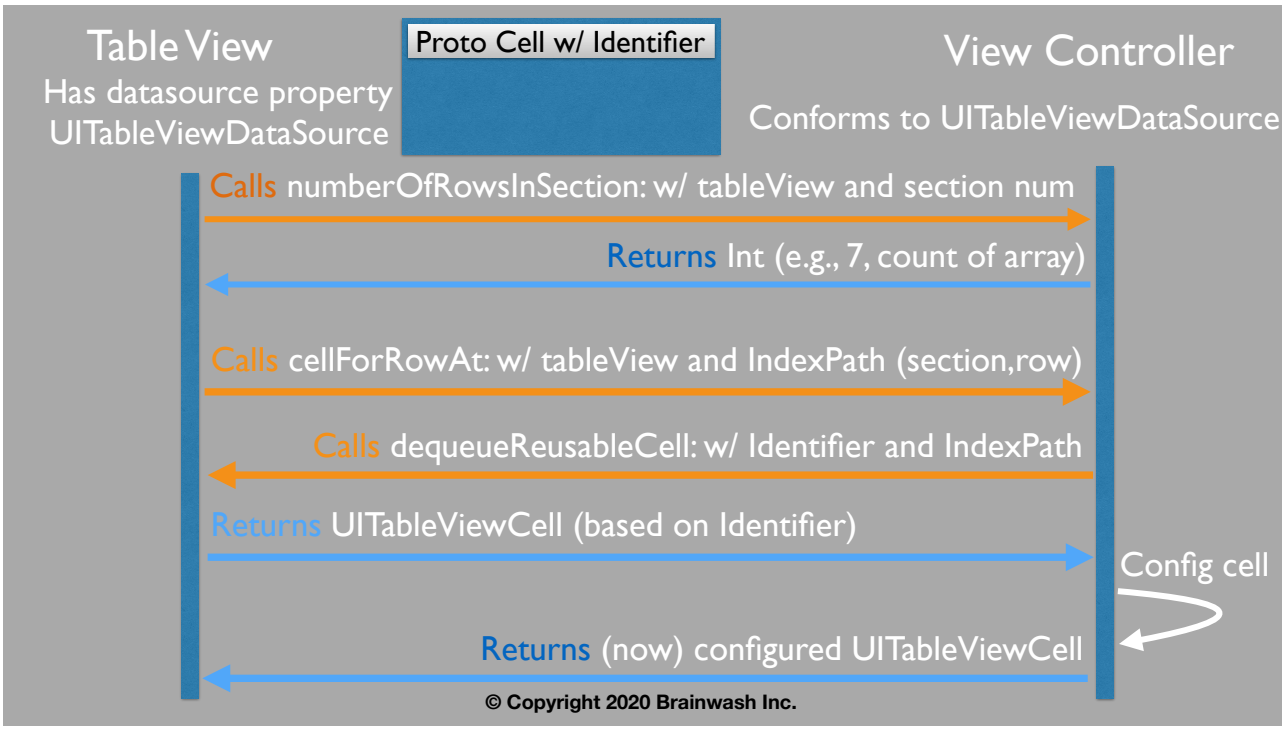


Table View
Table

Typically the View Controller with the Table View is set as the Datasource and Delegate

Required by UITableViewDataSource Protocol: number of rows, cell for row at index path (number of sections defaults to 1)

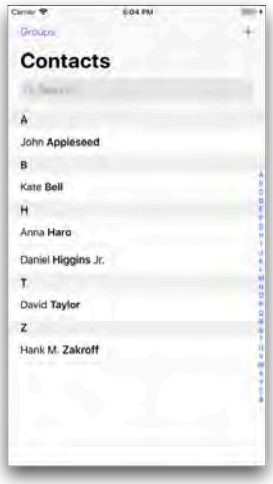
UITableViewDelegate Protocol: functions for display (height, will highlight, etc.) and action (did select, can edit, commit edit)




# Table View

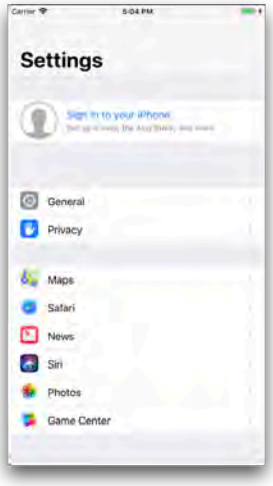
## Table

1 or more sections

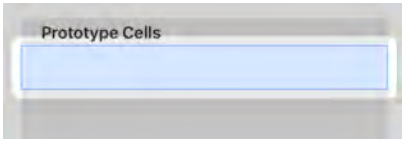



Each section can have header and/or footer (text or view)

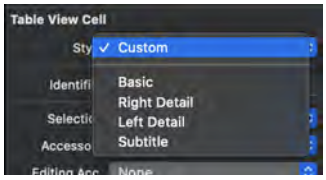
Sections may be plain or grouped



# TableView Cells



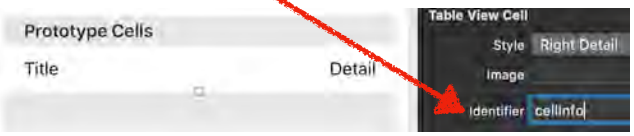

Prototype cells are defined in IB




A TableView can have multiple prototypes

Predefined cell styles

Cells need Identifiers to access from code






# TableView Cells

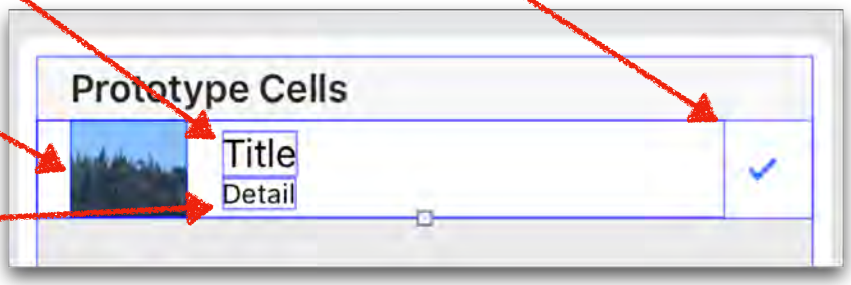
Cells have…

Text Label

Accessory View

Image View

Detail Label



TableView
Cells

Table View is designed with the prototype cell

Table View manages creating/recycling cells

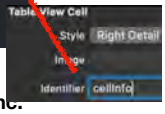
Table View is asked for a cell with Identifier (IB)

Your code configures the UI of the cell
  Recycled cells will have previous configuration



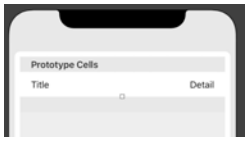
TableView
Cells

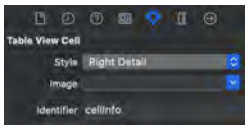
```
class ViewController: UIViewController,
        UITableViewDataSource, UITableViewDelegate {
    func numberOfSections(in tableView: UITableView) -> Int {
        return 5
    }

    func tableView(_ tableView: UITableView,
        numberOfRowsInSection section: Int) -> Int {
        return 10
    }

    func tableView(_ tableView: UITableView,
        titleForHeaderInSection section: Int) -> String? {
        return "Section \(section)"
    }

    func tableView(_ tableView: UITableView,
                   cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "cellInfo",
                                                 for: indexPath)
        cell.textLabel?.text = "\(Date())"
        cell.detailTextLabel?.text = "\(indexPath)"
        return cell
    }
```

# Lab 14: Table View

1. Create a new, single view project named LabTableV
2. Open Main.storyboard
3. Add a TableView from the Object Library to the View
4. Add a prototype cell from the Object Library to the table view
5. Set the cells Style to be “Right Detail”
6. Set the cells Identifier to be “cellInfo”
7. Set the Table View’s delegate and datasource outlets to be the ViewController





Continue…

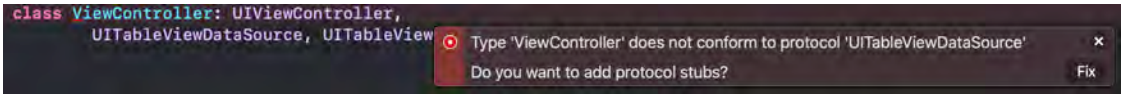


# Lab 14: Table View

8. In the ViewController.swift file, claim that the class implements the UITableViewDataSource and UITableViewDelegate protocols.

```
class ViewController: UIViewController,
        UITableViewDataSource, UITableViewDelegate {
```

9. Wait a moment until the error comes up and click on the red dot.




10. Click “Fix” to have the stubs added.
11. Return 5 from the numberOfSections function.

Continue…

## Lab 14: Table View

12. Implement cellForRowAt to…
    1. Call dequeueReusableCell with the Identifier from Interface Builder (“cellInfo”) and the indexPath passed in.
    2. Set the cell.textLabel.text to `"\(Date())"`
    3. Sell the cell.detailTextLabel.text to `"\(indexPath)"`
    4. Return the cell from the func
13. Optionally implement numberOfSections and titleForHeaderInSection (and other funcs from the UITableViewDataSource/Delegate protocols.
14. Run the app and verify results with scrolling.



## More TableView Delete


## Data Source protocol funcs

```
func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {
    return indexPath.section == 0 // only section 0 can be edited
}
func tableView(_ tableView: UITableView, titleForDeleteConfirmationButtonForRowAt
    indexPath: IndexPath) -> String? {
    return "Destroy"
}
func tableView(_ tableView: UITableView, commit
    editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
    switch editingStyle {
    case .delete:
        items.remove(at: indexPath.row)
        tableView.deleteRows(at: [indexPath], with: .automatic)
    default:
        print ("something else")
    }
}
```

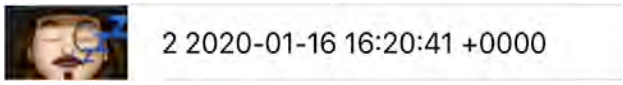
# More TableView Edit Actions


## iOS 8..<13 (deprecated), supersedes Delete title

```
// iOS 8+
func tableView(_ tableView: UITableView,
               editActionsForRowAt indexPath: IndexPath) -> [UITableViewRowAction]? {
    let actionDelete = UITableViewRowAction.init(style: .destructive,
                                                 title: "Trash") { (action, ip) in
        // delete item
        self.items.remove(at: ip.row)
        tableView.deleteRows(at: [indexPath], with: .automatic)
    }
    return [actionDelete]
}
```



# More TableView Swipe Actions




## iOS 11+, supersedes Delete title and Edit Actions

```
// iOS 11+
func tableView(_ tableView: UITableView, leadingSwipeActionsConfigurationForRowAt indexPath: IndexPath) ->
    UISwipeActionsConfiguration? {
    let actionSnooze = UIContextualAction.init(style: .normal,
                                               title: "Snooze")
    { (action, view, handler: (Bool)->Void) in
        // do something
        handler(true)
    }
    actionSnooze.image = UIImage.init(named: "snooze.png")
    let config = UISwipeActionsConfiguration.init(actions: [actionSnooze])
    return config
}
```

See Lab14-Extra.zip for examples

## Advanced TableView
## Refresh Control

View Controller can create UIRefreshControl

```
var refreshControl : UIRefreshControl!
```

Must set it's action, add to Table View and implement action func

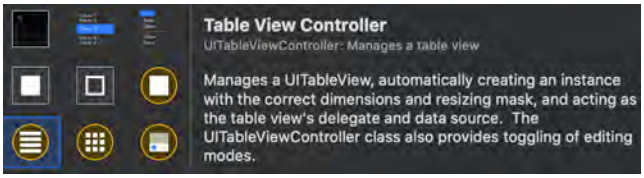
```
if tableView.refreshControl == nil {
    refreshControl = UIRefreshControl()
    refreshControl.addTarget(self,
        action: #selector(ViewController.didActivatePullToRefresh),
        for: .valueChanged)
    tableView.refreshControl = refreshControl
}
```

```
@objc func didActivatePullToRefresh() {
    refreshControl?.endRefreshing()
    // refresh table view
}
```

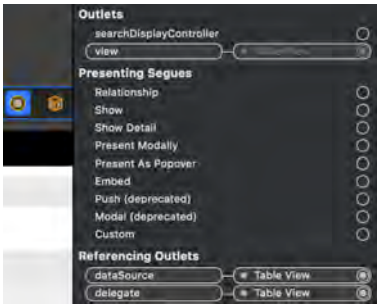
See Lab14-Extra.zip for example



## Advanced TableView
## Table View Controller



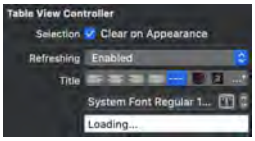

Inherits from UIViewController




Has tableView property - can be Static

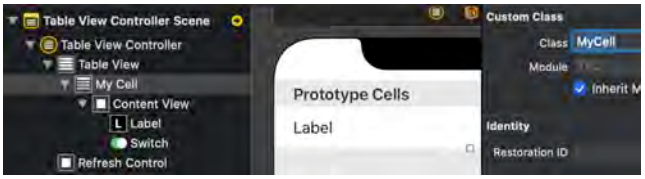
View property is also set to tableView

Implements UITableViewDataSource/Delegate



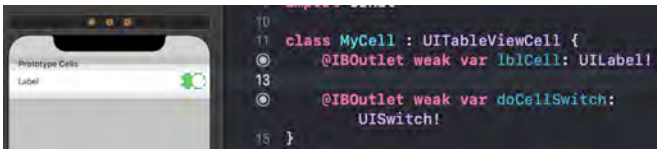

IB sets this up for you and provides Refresh Control

Advanced TableView
Custom Cells




Allows for… custom cells
  Add various UI elements

Must be instance of UITableViewCell subclass
  Allows for outlets and actions




The Cell might not be the best place to actually handle the actions - pass off to delegate.



Controllers
Navigation Controller

Stack of View Controllers
  Each VC is create when presented
  Added to the stack
  Released when navigated back (to previous)

Navigation Controller has no UI - it manages the stack and navigation to/from

It displays the “rootViewController”

Controllers
Segues

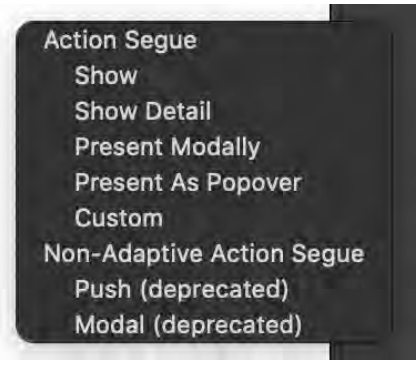
Navigational relationships between VCs

Adapt to current device and navigation
  Slide in for Navigation Controller
  Display modally without Navigation Controller

Control drag in IB (from button, cell, etc.) to another VC to create



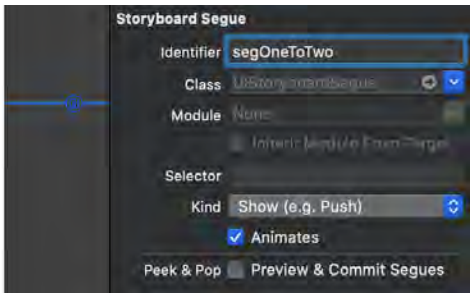

Controllers
Segues




Select Action Segue
  Show - typical
  Show Detail - see Master-Detail template
  Present Modally - modal even if Nav Controller
  Present as Popover - modal for iPhone
  Custom - your code handles transition

Controllers
Segues

Select Segue to set values
like Identifier




Use identifier to access in code

```
override func shouldPerformSegue(withIdentifier identifier:
    String, sender: Any?) -> Bool {
    if identifier == "segOneToTwo" {
        return false
    }
    return true
}
```

ShouldPerform - decide if the app should perform the segue/nav



Controllers
Segues

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

Another func - “prepare for segue”

Segue passed in - has properties for…
identifier - set in IB (e.g., segOneToTwo)
destination (view controller) - VC navigating to

Controllers
Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "segOneToTwo" {
        if let vc = segue.destination as? SecondViewController {
            vc.title = "\(Date())"
        }
    }
}
```

Destination VC doesn't have UI created yet - Don't access UI elements, only set properties

Cast it to the class it actually is - otherwise UIViewController



Controllers
Tab Bar

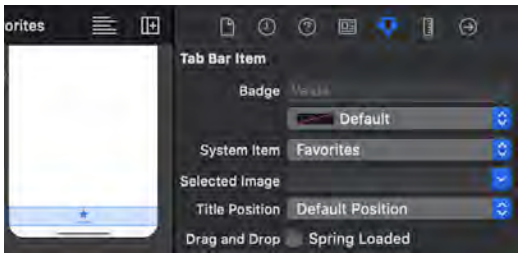
UITabBarController has a tab bar

One item per View Controller embedded


Controllers
Tab Bar

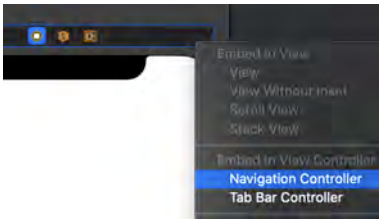
Selecting the item displays the VC

Items can be customized






## Lab 15: Navigation Controller

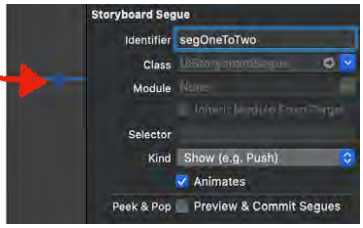



1. Create a new, single view project named LabNav
2. In Main.storyboard, select the View Controller
3. Select Embed In > Navigation Controller
    1. Notice the Navigation Controller is now the initial view controller
    2. It's relationship to the old VC is "rootViewController" - it will display first
4. Add UI elements to your View Controller.
5. Run and verify.

Continue…

# Lab 15: Navigation Controller


6. Add a second VC to your StoryBoard
7. Add a button to your first VC
8. Control+drag the button to your second VC
9. Select “Show” from the popup list of segue options
10. Click on the segue and view it’s attributes
11. Set the Identifier to segOneToTwo




Continue…



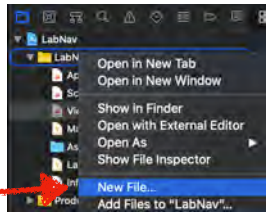
# Lab 15: Navigation Controller

13. In ViewController.swift “prepare for segue” to…
    1. Check the segue.identifier
    2. Get the destination view controller from the segue
    3. Set the title on the destination view controller to the date
14. Run and verify

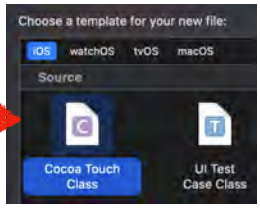
```
override func prepare(for segue: UIStoryboardSegue, sender:
    Any?) {
    if segue.identifier == "segOneToTwo" {
        let vc = segue.destination
        vc.title = "\(Date())"
    }
}
```

Continue…

# Lab 15: Navigation Controller




15. Add a file to your project
    1. Cocoa Touch Class
    2. Named SecondViewController
    3. Subclass of UIViewController
    4. Saved in the default project directory

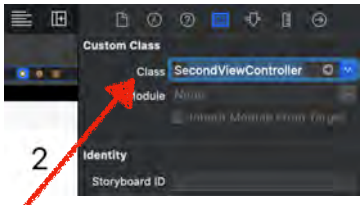




Continue…



# Lab 15: Navigation Controller




16. In Main.storyboard, made the second view controller and instance of your new SecondViewController class
17. Add a label to the UI
18. Create and outlet to it in SecondViewController.swift
19. And add a property “var text” as a String
20. In viewDidLoad set the lblSecond.text = self.text

Continue…

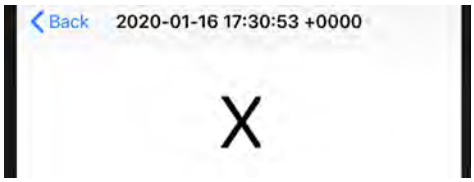
```
override func
    viewDidLoad() {
    super.viewDidLoad()
    lblSecond.text =
        self.text
```

```
    @IBOutlet weak var
        lblSecond: UILabel!
14  var text = ""
```

# Lab 15: Navigation Controller

21. Back in ViewController.swift, update “prepare for segue”…
    1. Optionally bind the destination VC to a SecondViewController
    2. Set the text property on the SecondViewController instance
22. Run and verify “X” is displayed.

```
if segue.identifier == "segOneToTwo" {
    if let vc = segue.destination as? SecondViewController {
        vc.title = "\(Date())"
        vc.text = "X"
    }
}
```






Data
Data Class

Commonly used in transmitting/reading/writing binary
Files (read/write), server (query/fetch)

Can be created by (and used to create) various instances:
String
UIImage
JSON/instances

Data
Data Class

String to-from Data round trip

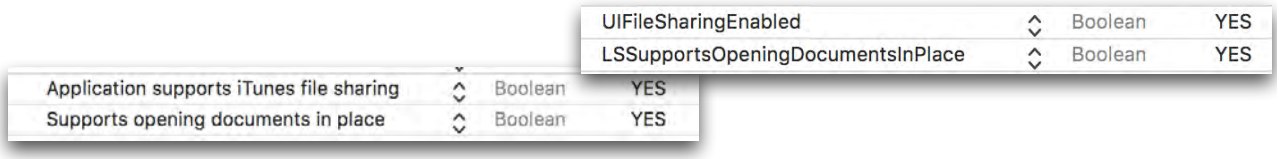
```
let stringVal = "String content"
let dataVal = stringVal.data(using: String.Encoding.utf8)
let backToString = String(data: dataVal!, encoding: String.Encoding.utf8)
print (backToString ?? "failed")
```



Data
Files

```
let documents = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first!
```

Directories
  Document - persists, back-up, shareable



Data
Files

Directories

```
let support = FileManager.default.urls(for: .applicationSupportDirectory, in: .userDomainMask).first!
```

```
let cache = FileManager.default.urls(for: .cachesDirectory, in: .userDomainMask).first!
```

```
let tempDir = NSTemporaryDirectory()
```



Data
Files

```
let fileURL = URL.init(fileURLWithPath: path)
guard let data = text.data(using: .utf8) else { return }
try? data.write(to: fileURL)
```

Data instances can write to files

- Get the path as a String
- Create a file URL
- Create a Data instance with data
- Call write on the Data instance

Data
Files

```
let fileURL = URL.init(fileURLWithPath: path)
guard let data = try? Data(contentsOf: fileURL) else { return "" }
return String(data: data, encoding: .utf8) ?? ""
```

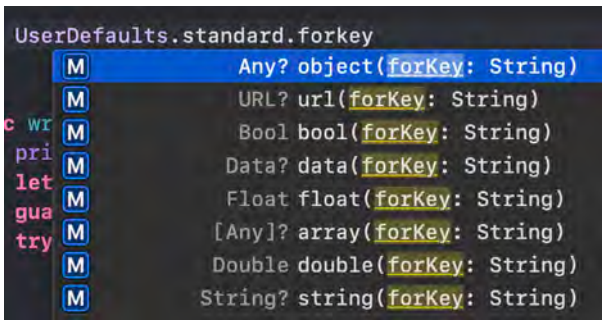
Data instances can read from files

- Get the path as a String
- Create a file URL
- Create a Data instance with file data
- Convert to usable object (e.g., image, string)

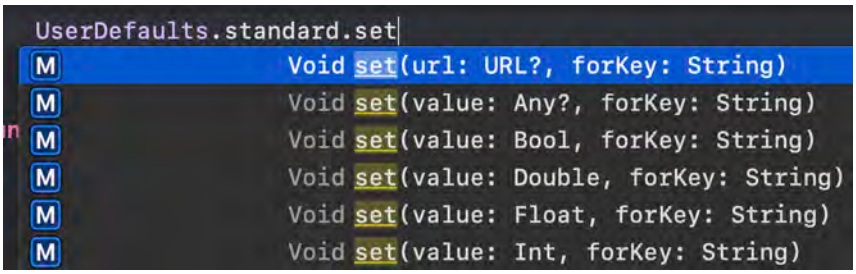


Data
UserDefaults

Persistent data stored by the device meant for small pieces of data (e.g., user settings)




Key-Value pairs



## Lab 16: Files

1. Create a new project called LabFiles
2. Write a func called writeFile that takes 2 params:
    1. text : String - a String to write to a file
    2. path : String - a String of the file path to write
3. Write a second func called readFile that returns a String (data read) and takes 1 param:
    1. path : String - a String of the file path to read
4. Call the functions to create a round trip like this:

```
let path = NSTemporaryDirectory() + "test.txt"
writeFile(text: "test", path: path)
let origText = readFile(path: path)
print (origText)
```



## Closures
## Higher Order Functions

Closures
Higher Order Functions

```
Void forEach(body: (String) throws -> Void)
```

foreach - parameter is a closure to call

The closure is called once for each item in the collection being iterated and that item is passed into the closure.



Closures
Higher Order Functions

```
Void forEach(body: (String) throws -> Void)
```

foreach

```
var names = ["Denton", "Mandeville", "Goteborg", "Bear", "Ventura"]
```

```
names.forEach({ (name) in
    print (name.count)
})
```

OR

```
names.forEach { (name) in
    print (name.count)
}
```

Output:

```
6
10
8
4
7
```

Closures
Higher Order Functions

```
names.forEach(body: (String) throws -> Void)
```

```
names.forEach { (String) in
    code
}
```

```
names.forEach { (name) in
    code
}
```



Closures
Higher Order Functions

```
[T] map(transform: (String) throws -> T)
```

map - Maps one collection to another (returns): same size, (possibly) different types (T Generic)

Type of return Collection is base on type closure returns

Map an Array of employ numbers to an Array (same size) of employ years of service.
Map an Array of names to an Array (same size) of lengths.

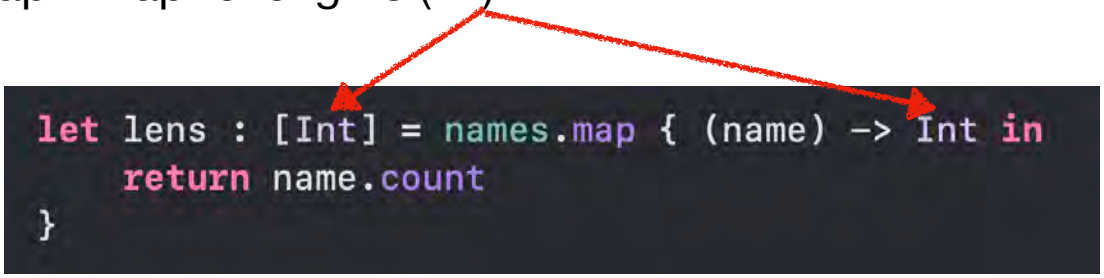
Closures

```
[T] map(transform: (String) throws -> T)
```

Higher Order Functions

```
var names = ["Denton", "Mandeville", "Goteborg", "Bear", "Ventura"]
```

map - Map to lengths (Int)



```
let lens : [Int] = names.map { (name) -> Int in
    return name.count
}
```

Result: `[6, 10, 8, 4, 7]`



Closures

```
[String] filter(isIncluded: (String) throws -> Bool)
```

Higher Order Functions

filter - Returns a collection of the same type, (possibly) smaller size (filters some out)

Parameter name (isIncluded) is hint: Closure should return true if the collection item “is included” (i.e., not filtered out).

Closures

```
[String] filter(isIncluded: (String) throws -> Bool)
```

Higher Order Functions

```
var names = ["Denton", "Mandeville", "Goteborg", "Bear", "Ventura"]
```

filter - Return true (isIncluded) for names > 6 chars

```
let longerNames = names.filter { (name) -> Bool in
    return name.count > 6
}
```

Result: **["Mandeville", "Goteborg", "Ventura"]**



```
[ElementOfResult] compactMap(transform: (String) throws -> ElementOfResult?)
```

Closures

Higher Order Functions

compactMap - Like map and filter combined

ElementOfResult is the Generic
possibly diff type

Optional - if nil, it's filtered out
possibly diff size

```
[ElementOfResult] compactMap(transform: (String) throws -> ElementOfResult?)
```

Closures
Higher Order Functions

```
var names = ["Denton", "Mandeville", "Goteborg", "Bear", "Ventura"]
```

compactMap

```
let longerLens : [Int] = names.compactMap { (name) -> Int? in
    return name.count > 6 ? name.count : nil
}
```

Result: `[10, 8, 7]`



```
Result reduce(initialResult: Result, nextPartialResult: (Result, String) throws -> Result)
```

Closures
Higher Order Functions

reduce - e pluribus enum: out of man, one

Take a collection and “reduce” it to one value:
- Concatenate
- Sum
- Join
- Find
- Compare

```
Result reduce(initialResult: Result, nextPartialResult: (Result, String) throws -> Result)
```

Closures
Higher Order Functions

reduce - Params:
initialResult - the start value: 0, “”, “Names: “
  Type: Same as ultimate Result and 1st param of closure

nextPartialResult - closure to get the ongoing result: sum, concatenation, join, etc.
  Result - ongoing value
  String - (based on collection), next collection item
  Returns - ongoing value for next iteration, final call = final result



```
Result reduce(initialResult: Result, nextPartialResult: (Result, String) throws -> Result)
```

```
let oneVal = names.reduce("") { (ongoing, next) -> String in
    print ("---\nongoing: \(ongoing)\nNext: \(next)\nReturning: \(ongoing + next + " ")")
    return ongoing + next + " "
}
```

```
---
ongoing:
Next: Denton
Returning: Denton
---
ongoing: Denton
Next: Mandeville
Returning: Denton Mandeville
---
ongoing: Denton Mandeville
Next: Goteborg
Returning: Denton Mandeville Goteborg
---
ongoing: Denton Mandeville Goteborg
Next: Bear
Returning: Denton Mandeville Goteborg Bear
---
ongoing: Denton Mandeville Goteborg Bear
Next: Ventura
Returning: Denton Mandeville Goteborg Bear Ventura
Denton Mandeville Goteborg Bear Ventura
```

© Copyright 2020 Brainwash Inc.

```
Result reduce(initialResult: Result, nextPartialResult: (Result, String) throws -> Result)
```

```
let anotherVal = names.reduce(0) { (ongoing, next) -> Int in
    print ("---\nongoing: \(ongoing)\nNext: \(next)\nReturning: \(ongoing + next.count)")
    return ongoing + next.count
}
```




Closures
Higher Order Functions

contains - custom find to return boolean

drop - closure returns true for items to be dropped, once it returns false, the rest are in result

first - custom closure for condition of finding first item

## Closures
## Higher Order Functions

forEach - execute the closure once for each item in collection

partition - inline change of collection into items that closure returned false, then items for true

split - creates 2+ collections based on separator (true from closure)



## Closures
## Closures

Closures have types - parameter for forEach:

```
Void forEach(body: (String) throws -> Void)
```

The body parameter type is: (String) -> Void (throws possible)

## Closures
## Closures

Closures have types - parameter for forEach:

```
Void forEach(body: (String) throws -> Void)
```

The body parameter type is: (String) -> Void (throws possible)

So we could have variable of that type and value:

```
let closureToPrint : (String)->Void = { (name:String) in
    print (name)
}
```



## Closures
## Closures

So we could have variable of that type and value:

```
let closureToPrint : (String)->Void = { (name:String) in
    print (name)
}
```

So we can call forEach with that variable:

```
names.forEach(closureToPrint)
```

Closures
Function Types

Functions are closures with names
  So functions have types (just like closures… b/c they are)

Functions can be…
  Assigned to variables
  Passed into funcs
  Returns from funcs

Common for threaded callback completion handlers (coming)



## Lab 17: Higher Order Functions

```
var names = ["Denton", "Mandeville", "Goteborg", "Bear", "Ventura"]
```

1. Create a new project called LabHOF
2. In the ViewController code, create an array similar to the one above
3. Write functions to…
    1. Use forEach to print each Array element
    2. Use map to get an Array of the element lengths
    3. Use filter to get an Array of only names > 6 chars long
    4. Do step 2 and 3 in one call to compactMap
    5. Use reduce to concatenate all the Array elements
    6. Use reduce to sum all the Array elements' lengths

See Lab17.zip for help/solution.

## Server Communication
## URLSession

URLSession - create tasks to hit a URL
  Can download to memory (Data instance) or file (URL)
  File downloads are to temp directory - must be moved

Create tasks with URL for GETs
  To start task, call resume() func

Create tasks with URLRequest for POSTs
  URLRequest can set headers, body, etc.



## Server Communication
## URLSession

URLRequest

```
var urlReq = URLRequest.init(url: URL.init(string: fromURL)!,
                             cachePolicy: .reloadRevalidatingCacheData,
                             timeoutInterval: 10.0)
urlReq.httpMethod = "POST"
urlReq.httpBody = "{\"name\":\"test\"}".data(using: .utf8)
urlReq.addValue("text/json", forHTTPHeaderField: "Content-Type")
```

Server Communication
URLSession

URLSession -

Second parameter is a closure called after data is downloaded
- Background thread
- UI updates must be on main/UI thread



Server Communication
URLSession

Data can then be converted to String, Image, etc.

```
if let url = URL.init(string: fromURL) {
    let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
        // check error, data
        guard error == nil, data != nil else { return }
        print (String.init(data: data!, encoding: .ascii) ?? "no value")
    }
    task.resume()
}
```

# Server Communication
# URLSession

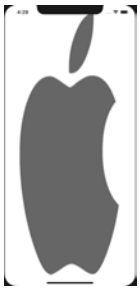
## Fetch Data, create image, update UI

```
if let url = URL.init(string:
    "https://www.apple.com/ac/structured-data/images/knowledge_graph_logo.png?201606271147") {
    let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
        DispatchQueue.main.async {
            let iv = UIImageView.init(frame: self.view.bounds)
            iv.image = UIImage.init(data: data!)
            self.view.addSubview(iv)
        }
    }
    task.resume()
}
```



# Lab 18: Server Communication

1. Create a new project called LabURL
2. Write a function that uses URLSession to create a task using…
    1. URL: https://apple.co/2FZdO9A
    2. A closure taking data, response, error
3. Start a closure on the main thread with: DispatchQueue.main.async
4. Create a UIImageView with the same frame as the self.view.bounds
5. Create an Image with the Data instance downloaded
6. Set the created image as the UIImageView image (property)
7. Add the UIImageView to the view via self.view.addSubview



See Lab18.zip for help/solution.

JSON & Codable

RESTful JSON APIs are common, readable and easily map to Arrays/Dictionaries

So let's have the objects do the work: Codable

It's really two protocols…

```
/// `Codable` is a type alias for the `Encodable` and `Decodable` protocols.
/// When you use `Codable` as a type or a generic constraint, it matches
/// any type that conforms to both protocols.
public typealias Codable = Decodable & Encodable
```



JSON & Codable

More good news…

If your properties names/types match the JSON, done!

```
{
    "id": "1",
    "music_url": "http://orangevalleycaa.org/api/media/music
    /ModernHardRock_45450.wav",
    "name": "Modern Hard Rock",
    "description": "Powerful rock track",
    "image": "http://orangevalleycaa.org/api/media/images/
    ArtistWorking_059730538.png",
    "thumb": "http://orangevalleycaa.org/api/media/images/
    thumbs/ArtistWorking_059730538.png",
}
```

```
struct Music : Codable {
    var id = ""
    var music_url : URL?
    var name = ""
    var description = ""
    var image : URL?
    var thumb : URL?
}
```

JSON & Codable
Decodable

Creates a new instance by decoding from the given decoder

JSONDecoder is provided!

```
/// - Parameter decoder: The decoder to read data from.
init(from decoder: Decoder) throws
```



JSON & Codable
Decodable

```
let items = try? JSONDecoder().decode([Music].self, from: data)
```

Create a JSONDecoder
Call decode with…
  Type you're expecting: <Type>.self or similar
  Data
decode throws so use try? (or wrap in do-try-catch)

Data - We've seen how to get Data instances from Strings, files and server queries

JSON & Codable
Decodable
Given JSON text or Data


We can decode to Music instances

```swift
func loadMusic() -> [Music]? {
    guard let data = jsonString.data(using: .utf8) else { return nil }
    return decodeJSON(data: data)
}

func decodeJSON(data : Data) -> [Music]? {
    let items = try? JSONDecoder().decode([Music].self, from: data)
    return items
}
```



JSON & Codable
Encodable
Encodes this value into the given encoder

JSONEncoder is also provided!

```swift
/// - Parameter encoder: The encoder to write data to.
func encode(to encoder: Encoder) throws
```

```swift
func encodeToJSON(music : [Music]?) -> Data? {
    guard let music = music else { return nil }
    guard let data = try? JSONEncoder().encode(music) else { return nil }
    return data
}
```

JSON & Codable
Encodable

JSONEncoder is also provided!

```
/// - Parameter encoder: The encoder to write data to.
func encode(to encoder: Encoder) throws
```

```
func encodeToJSON(music : [Music]?) -> Data? {
    guard let music = music else { return nil }
    guard let data = try? JSONEncoder().encode(music) else { return nil }
    return data
}
```



JSON & Codable
CodingKeys

JSON-Properties mismatch
Create CodingKeys enum
  String underlying type
  Conforms to CodingKey

Any properties not in enum
  are ignored both directions

```
struct Music : Codable {
    var id = ""
    var music_url : URL?
    var name = ""
    var description = ""
    var image_url : URL?
    var thumb_url : URL?
    var dirty = false

    enum CodingKeys: String, CodingKey {
        case id = "id"
        case music_url = "music_url"
        case name = "name"
        case description = "description"
        case image_url = "image"
        case thumb_url = "thumb"
    }
}
```

# JSON & Codable Protocols

You can override the encode and init funcs…

But then what's the point?

```
func encode(to encoder: Encoder) throws {
    var container = encoder.container(keyedBy: CodingKeys.self)
    try container.encode(name.replacingOccurrences(of: "name: ", with: ""),
                         forKey: .name)
    try container.encode(name, forKey: .name)
}

init(from decoder: Decoder) throws {
    if let container = try? decoder.container(keyedBy: CodingKeys.self) {
        if let val = try? container.decode(String.self, forKey: .name) {
            self.name = "name: \(val)"
        }
        if let val = try? container.decode(String.self, forKey: .id) {
            self.id = val
        }
    }
}
```



# Lab 19: Server/JSON

1. Create a new project called LabJSON
2. In ViewController.swift create a Music class (or struct)
3. Have Music conform to Codable & have properties…
4. Add a property to ViewController to hold an array of Music items: [Music]
5. Write a function that uses URLSession to create a task using…
    1. URL: https://orangevalleycaa.org/api/music
    2. A closure taking data, response, error
6. Parse the returned Data instance to an Array of Music items using JSONDecoder
7. Run and verify data is fetched and decoded.

```
struct Music : Codable {
    var id = ""
    var music_url : URL?
    var name = ""
    var description = ""
    var image : URL?
    var thumb : URL?
}
```

- Bonus: Write the fetched data to a file and use it as a cache if the app is run within 5 minutes of when the data was last fetched (you can store the previous fetch time in UserDefaults).

See Lab19.zip for help/solution. Includes String parsing of JSON and CodingKeys

## UI Interaction
## Touches

```
M touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?)
M touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?)
M touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)
M touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?)
Tells this object that one or more new touches occurred in a view or window.
```

Detect touches on any View

Four functions to override
- Passed in a Set (no order) of touches (max: 5)
- Event

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    guard let point = touches.first?.location(in: self.view) else { return }
    let v = UIView(frame: CGRect(origin: point, size: CGSize(width: 10, height: 10)))
    v.backgroundColor = UIColor.purple
    self.view.addSubview(v)
}
```

Can get point (x,y) of touch in view



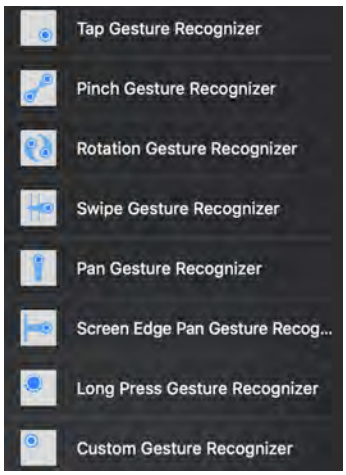
## UI Interaction
## Gestures

Gestures

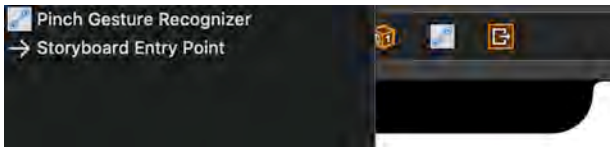
Various types

Drop on UI (or create in code)

Attach event/action in code



UI Interaction
Gestures




Gestures

Passed in Gesture instance to get info: pinch-scale, rotation angle, etc.

```
@IBAction func handlePInch(_ sender: UIPinchGestureRecognizer) {
    let angle = sender.scale
    print (angle)
}
```



UI Interaction
Animation

UIView animation funcs

```
Void animate(withDuration: TimeInterval, animations: () -> Void)
Void animate(withDuration: TimeInterval, animations: () -> Void, completion: ((Bool) -> Void)?)
Void animate(withDuration: TimeInterval, delay: TimeIn…tions: () -> Void, completion: ((Bool) -> Void)?)
Void animateKeyframes(withDuration: TimeInterval, dela…tions: () -> Void, completion: ((Bool) -> Void)?)
Void animate(withDuration: TimeInterval, delay: TimeIn…tions: () -> Void, completion: ((Bool) -> Void)?)
```

Put UI changes in the “animations" parameter (closure)

Changes are extrapolated over the duration

# UI Interaction
# Animation

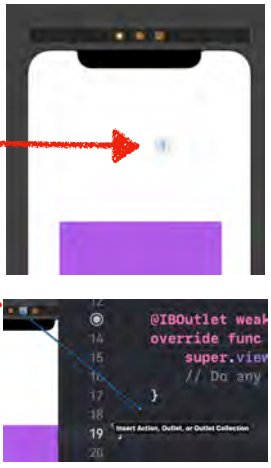
```
UIView.animate(withDuration: 3.0) {
    self.vBox.center = point
    self.vBox.backgroundColor = UIColor.blue
    self.vBox.layer.cornerRadius = self.vBox.frame.size.height/2
}
```

Over 3 seconds…

- Move the vBox View's center to a new point
- Change the background color to blue
- Round the corners to have the height



# Lab 20: Gestures/Animation



1. Open LabAnim from Lab20.zip (provided)
2. In Main.storyboard, add a Pan gesture to the main View (not the box)
3. Create an action from the gesture to the ViewController code
4. Name the Action handlePan
5. Set the Type UIPanGestureRecognizer




6. In the action func, get the point in the view the user is touching and use that to change the center of the vBox view:

```
@IBAction func handlePan(_ sender: UIPanGestureRecognizer) {
    let point = sender.location(in: self.view)
    vBox.center = point
}
```

Continue…

# Lab 20: Gestures/Animation

7. Animate the change (and possibly others)

```
UIView.animate(withDuration: 3.0) {
    self.vBox.center = point
    self.vBox.backgroundColor = UIColor.blue
    self.vBox.layer.cornerRadius = self.vBox.frame.size.height/2
}
```


- Bonus: Use other gestures and other animations.

See Lab20b.zip for help/solution.



# Notifications Local

```
func reqNotifAuth() {
    let center = UNUserNotificationCenter.current()
    center.requestAuthorization(options: [.alert, .sound]) { (granted, error) in
        guard error == nil, granted == true else { return }
        print ("granted")
    }
}
```

Request and handle authorization based on type: alert, badge, sound

Content: Create a content object and set properties: title, body

Trigger: Create a trigger: calendar, time interval or location

Create notification request with content and trigger

Add notification request to Notification Center

# Notifications
# Local

## Content

```
let content = UNMutableNotificationContent()
content.title = "Times up!"
content.body = "It's been 10 seconds."
content.sound = UNNotificationSound.default // or...
content.sound = UNNotificationSound.init(named: UNNotificationSoundName(rawValue: "mysound.wav"))
```

Title - text
Body - text
Sound - default or uncompressed audio compiled into app
Category - left/right swipe actions including foreground and background



# Notifications
# Local

## Trigger Types

Calendar (date) - repeat is based on granularity of the date: hour -> dailing

```
let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 10.0,
                                                     repeats: false)
var dateInfo = DateComponents()
dateInfo.hour = 11
dateInfo.minute = 30
```

Time Interval - seconds

```
let trigger = UNTimeIntervalNotificationTrigger.init(timeInterval: 10.0,
                                                     repeats: false)
var dateInfo = DateComponents()
dateInfo.hour = 11
dateInfo.minute = 30
```

# Notifications
# Local

## Trigger Types

```
let center = CLLocationCoordinate2D(latitude: 37.335400, longitude: -122.009201)
let region = CLCircularRegion(center: center, radius: 2000.0, identifier: "Headquarters")
region.notifyOnEntry = true
region.notifyOnExit = false
let t3 = UNLocationNotificationTrigger(region: region, repeats: false)
```

Location - lat/long, radius
  Requires CoreLocation and user authorization to use location
  Doesn't require “Always" because system monitors for you



# Notifications
# Local

## Create Request with Identifier

```
let req = UNNotificationRequest.init(identifier: "Timer",
                                     content: content, trigger: trigger)
```

## Add to Notification Center

```
UNUserNotificationCenter.current().add(req, withCompletionHandler: nil)
```

## Accessing current notifs

```
UNUserNotificationCenter.current().getPendingNotificationRequests { (notifs) in
    notifs.forEach({ (req) in
        print (req.content.body)
        UNUserNotificationCenter.current().removeAllPendingNotificationRequests()
    })
}
```

Notifications
Push

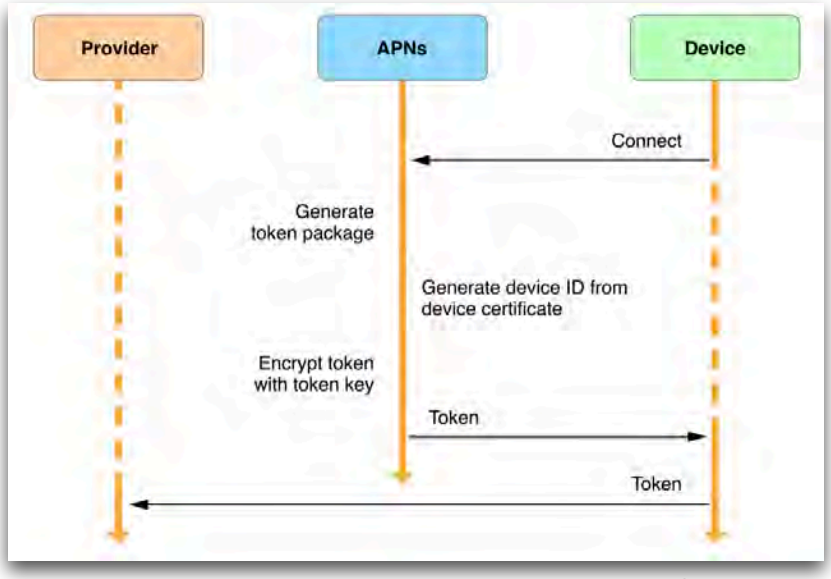
Same authorization as local

Notifs come from Apple (APNS)

3rd Party Services: UrbanAirship, Firebase, AWS

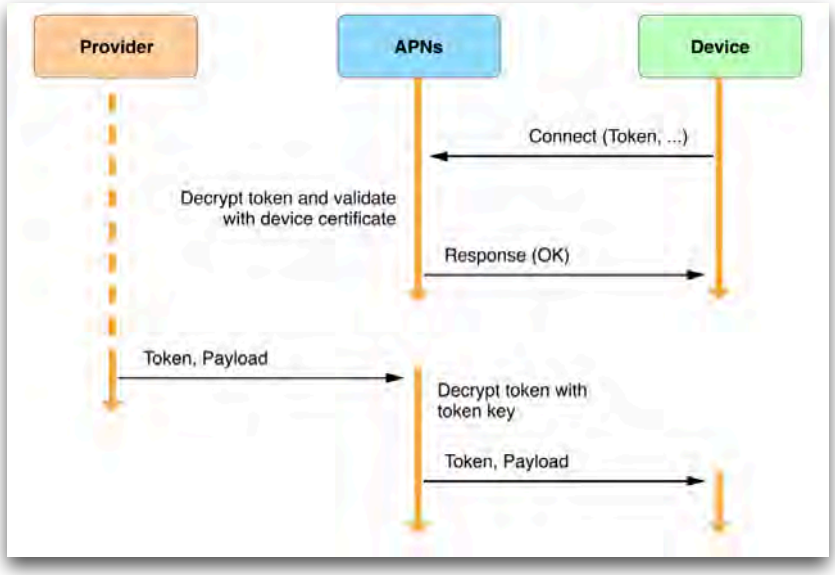


Notifications
Push




From https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html

# Notifications
# Push



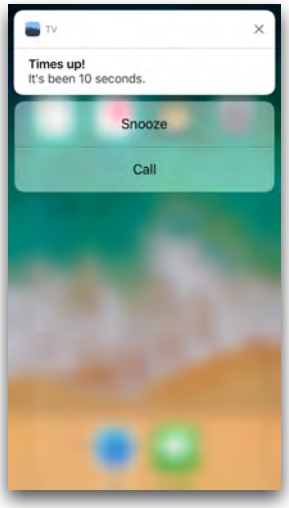

From https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html



# Notifications
# Push

User can control display options

Up to 10 actions can be displayed



Notifications
Handling Notifications

Handled in code

Set AppDelegate as
UNUserNotificationCenter.current().delegate

Implement UNUserNotificationCenterDelegate



Notifications
Handling Notifications

Two functions:
willPresent (foreground)

```
func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification,
    withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void) {
}
```

didRecieve (background)

```
func userNotificationCenter(_ center: UNUserNotificationCenter, didReceive response: UNNotificationResponse,
    withCompletionHandler completionHandler: @escaping () -> Void) {
    print(response.notification.request.content.categoryIdentifier)
    print(response.actionIdentifier)
}
```

Notifications
NotificationCenter

Very different - internal, no UI

Any class instance can broadcast a message

Any (other) instance can register to receive it

Can include other data to be passed to receiver's func



Notifications
NotificationCenter

Receiver needs to be added as observer for message

```
NotificationCenter.default.addObserver(self,
          selector: #selector(ViewController.handleNotif),
          name: NSNotification.Name(rawValue: "NewData"), object: nil)
```

Func then gets called passing in the notification

```
@objc func handleNotif(notif : Notification) {
    print (notif.userInfo)
}
```

Notifications
NotificationCenter

Posting the message

```
NotificationCenter.default.post(name: NSNotification.Name(rawValue: "NewData"),
                                object: self, userInfo: ["key1":"val1"])
```

Sent to all observers

Good for singletons needs to send out updates



Threading
GCD & UI

```
DispatchQueue.main.async {
    // main/UI thread
}
```

Grand Central Dispatch - manages threads

UI updates are done on the main/UI thread: sync/async

UI actions are on main thread
  Event actions don't need this block

Threading
Background

Global background thread
  Do processing
  Update UI on main

```
DispatchQueue.global().async {
    // background

    DispatchQueue.main.async {
        // main/UI thread
    }
}
```



Threading
Background

```
DispatchQueue(label: "background").async {
    for i in 0..<10 { print ("1 \(i)") }
}
DispatchQueue(label: "background2").async {
    for i in 0..<10 { print ("2 \(i)") }
}
```

```
1 0
2 0
2 1
2 2
2 3
2 4
2 5
1 1
1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9
2 6
2 7
2 8
2 9
```

Create background threads: async

2 threads
  Each go 0 to 9
  In order
  About the same time

Threading
Background

```
for i in 0..<10 {
    DispatchQueue(label: "background\(i)").async {
        print ("1 \(i)")
    }
}
for i in 0..<10 {
    DispatchQueue(label: "background2\(i)").async {
        print ("2 \(i)")
    }
}
```

```
1 3
1 7
1 1
1 9
1 4
2 0
1 6
2 1
1 8
2 2
1 2
1 0
1 5
2 4
2 3
2 5
2 6
2 7
2 8
2 9
```

Create background threads: async

20 threads
  Each prints one line
  About the same time



Threading
Background

```
for i in 0..<10 {
    DispatchQueue.global().sync {
        print ("1 \(i)")
        print ("2 \(i)")
    }
}
```

```
1 0
2 0
1 1
2 1
1 2
2 2
1 3
2 3
1 4
2 4
1 5
2 5
1 6
2 6
1 7
2 7
1 8
2 8
1 9
2 9
```

Create background threads: sync

10 threads
  Each prints two lines
  In order

Threading
Background

```
for i in 0..<10 {
    DispatchQueue.global().async {
        print ("1 \(i)")
        print ("2 \(i)")
    }
}
```

```
1 1
2 1
1 2
2 2
1 5
1 0
1 3
2 3
1 4
2 4
1 6
1 7
2 7
2 0
1 8
2 8
2 6
2 5
1 9
2 9
```

Create background threads: async

10 threads
  Each prints two lines
  Not ordered (but close, varies)



Frameworks
CoreData

Wrapper for database:
  SQLite (default), binary, XML or Memory

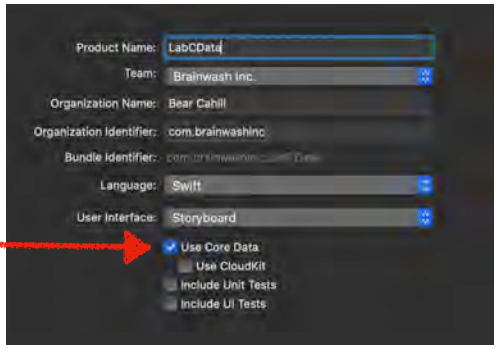
**Table 16-1** Built-in persistent store types

| Store type | Speed | Object graph in memory | Other factors |
|---|---|---|---|
| XML (atomic) | Slow | Whole | Externally parsable |
| Binary (atomic) | Fast | Whole | N/A |
| SQLite | Fast | Partial | N/A |
| In-memory | Fast | Whole | No on-disk storage required |

Image from https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/PersistentStoreFeatures.html

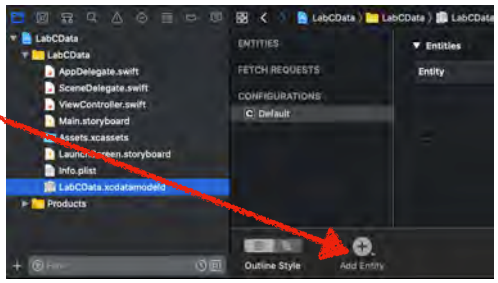
Frameworks
CoreData

Added when you create a project



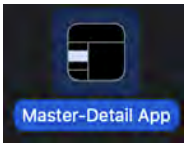

Creates empty Model

You can add Entities (tables)
With Attributes (columns)



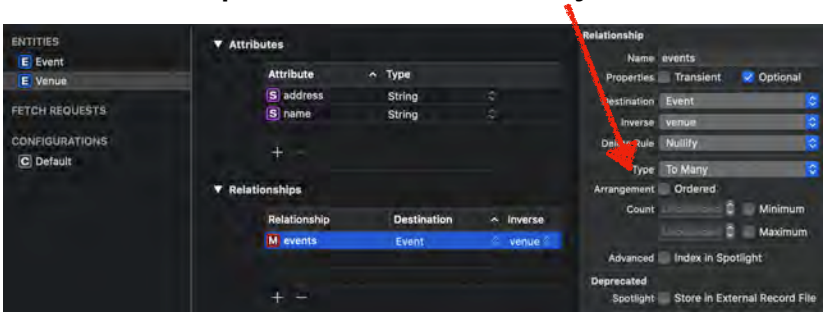


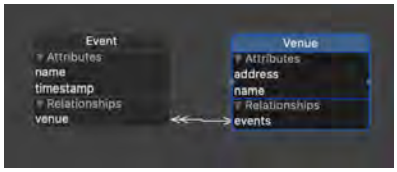

Frameworks
CoreData




Master-Detail template creates basic Model

Relationships can be 1-many



Frameworks
CoreData

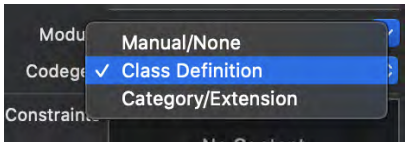
M-D template includes code for CoreData objects
Including creating new items in database

```
@objc
func insertNewObject(_ sender: Any) {
    let context = self.fetchedResultsController.managedObjectContext
    let newEvent = Event(context: context)

    // If appropriate, configure the new managed object.
    newEvent.timestamp = Date()

    // Save the context.
    do {
        try context.save()
    } catch {
```



Frameworks
CoreData

Codegen options:

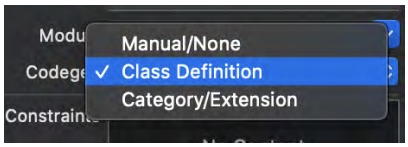



Manual/None
Can use Editor > Create NSManaged Subclass…
Generates Class and Extension (Properties, overwritten)

Class Definition (default)
Build generates and includes code (not visible, editable)

Frameworks
CoreData

Codegen options:




Category/Extension
- Generates Category (Objc) or Extension (not visible)
- Expects you to provide Class
- Can use Editor > Create NSManaged Subclass…
- Generates Class and Extension (Properties, overwritten)
- Delete Extension files



Frameworks
CoreLocation

System uses GPS, Cell tower triangulation and/or wifi address for best location data

Authorization from user is required: always, when in use

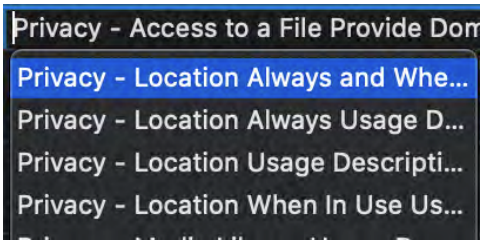
Background mode is available (see Capabilities)

Location manager can update location and heading

Location includes lat, long, altitude and accuracy

Filter and accuracy can be specified

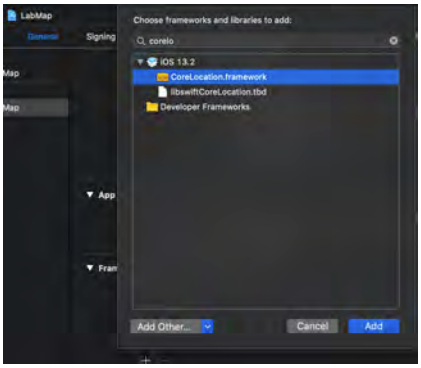
# Frameworks
# CoreLocation



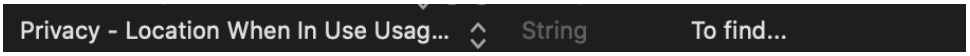

Add CoreLocation to your project

Add plist value: Privacy - Location

When In Use Usage Description (or NSLocationWhenInUseUsageDescription) - or “Always”




Import CoreLocation into code






# Frameworks
# CoreLocation

```
class ViewController: UIViewController, CLLocationManagerDelegate {

    var locMgr : CLLocationManager?

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        CLLocationManager.locationServicesEnabled()

        locMgr = CLLocationManager()
        locMgr?.delegate = self
        locMgr?.requestWhenInUseAuthorization()
```

Check if location services are enabled

Create Location Manager and set delegate (CLLocationManagerDelegate)
Request authorization: “when in use” authorization
Description value is included in request to user

# Frameworks
# CoreLocation


Request for Always includes "when in use"

Callback for authorization change

CLAuthorizationStatus: authorizedWhenInUse, authorizedAlways, denied, notDetermined (no response), restricted (restricted location services, user may not be admin)

```
func locationManager(_ manager: CLLocationManager,
                     didChangeAuthorization status: CLAuthorizationStatus) {
    if status == .authorizedWhenInUse {
    }
}
```



# Frameworks
# CoreLocation

```
if status == .authorizedWhenInUse {
    manager.desiredAccuracy = 100 // meters
    manager.distanceFilter = 100 // meters
    manager.startUpdatingLocation()
}
```

Set values on property or manager passed in and start updates

Accuracy - desired accuracy of the values returned, not guaranteed, larger number = less power

Filter distance - how far the device has to move before sending and update to the app

```
public let kCLLocationAccuracyBestForNavigation: CLLocationAccuracy
public let kCLLocationAccuracyBest: CLLocationAccuracy
public let kCLLocationAccuracyNearestTenMeters: CLLocationAccuracy
public let kCLLocationAccuracyHundredMeters: CLLocationAccuracy
public let kCLLocationAccuracyKilometer: CLLocationAccuracy
public let kCLLocationAccuracyThreeKilometers: CLLocationAccuracy
```

## Frameworks
## CoreLocation

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    if let loc = locations.last {
        print ("\(loc.coordinate.latitude) \(loc.coordinate.longitude) \(loc.course) \(loc.speed)")
    }
}
```

didUpdateLocations - parameter for array of locations (always at least 1), last is latest

CLLocation has properties for course  (degrees), speed (meters per second), accuracy (horiz and vertical)

distance function - meters from one location to another



## Frameworks
## CoreLocation

To stop: stopUpdatingLocation - stop when possible (power)

Separately the location manager can start updates for heading - start/stopUpdatingHeading

didUpdateHeading callback includes CLHeading instance (1) Heading has magnetic heading, true heading, timestamp

## Frameworks
## CoreLocation: Background

```
if CLLocationManager.significantLocationChangeMonitoringAvailable() {
    manager.startMonitoringSignificantLocationChanges()
}
```

If available, monitor Significant location changes in the background - start/stopMonitoringSignificantLocationChanges

Ignores any distance filter and accuracy settings

Requires “always” authorization

Same didUpdateLocations callback



## Frameworks
## CoreLocation: Background

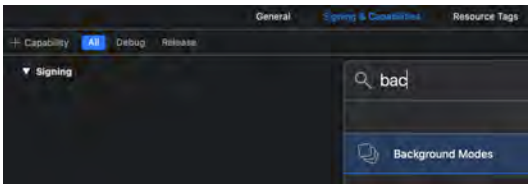
Will launch app if necessary

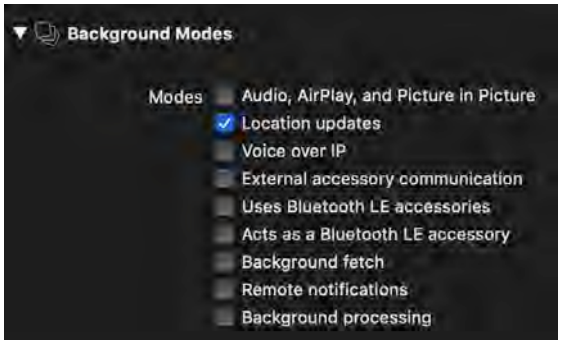
didFinishLaunchingWithOptions with location launch key set

Need to recreate/configure location manager and call startMonitoringSignificantLocationChanges

Be mindful of when your location manager delegate class is created

# Frameworks
# CoreLocation: Background




Background Modes (Signing & Capabilities > + Capability), required mode to Info.plist

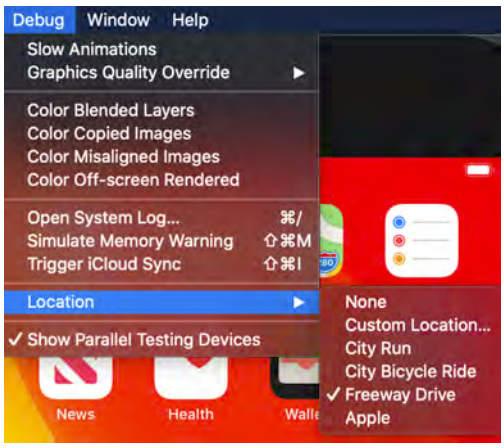
Tells system app requires access (and launching) in background

Set location manager allowsBackgroundLocationUpdates to true

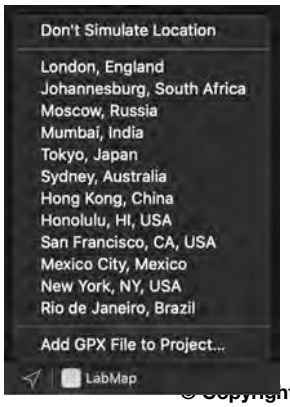


# Frameworks
# CoreLocation: Simulate

Simulator: Debug > Location




Xcode



## Frameworks
## CoreLocation: Perf&Power

```
locMgr?.pausesLocationUpdatesAutomatically = true
```

Use the highest accuracy and filter you can - or switch after obtaining location

Monitor significant location changes if possible

Allow the system to pause updates when the user is not likely to move (mgr.pausesLocationUpdatesAutomatically = true)

Don’t use “always” and background



## Frameworks
## CoreLocation: Reverse/Geocode

CoreLocation Geocoding and Reverse Geocoding

CLGeocoder.geocodeAddressString - takes String of address returns array of CLPlacemarks which contain location and other values

CLGeocoder.reverseGeocodeLocation - takes a CLLocation, returns array of CLPlacemarks

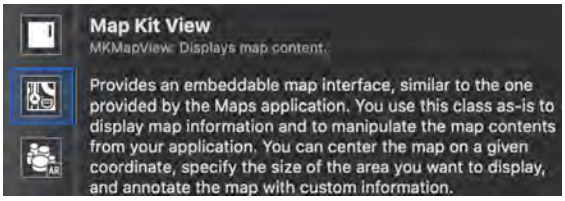
# Frameworks
# CoreLocation: Reverse/Geocode

```
func geocode() {
    CLGeocoder().geocodeAddressString("300 Oak Street, Denton, TX") {
        (placemarks, error) in
        for pm in placemarks! {
            print (pm.location?.coordinate.latitude ?? "no lat")
            print (pm.location?.coordinate.longitude ?? "no long")
            self.revgeocode(loc: pm.location!)
        }
    }
}

func revgeocode(loc : CLLocation) {
    CLGeocoder().reverseGeocodeLocation(loc) { (placemarks, error) in
        for pm in placemarks! {
            // CNPostal... requires Contacts Framework
            print (CNPostalAddressFormatter.string(from:pm.postalAddress!,
           style: .mailingAddress))
        }
    }
}
```

```
33.2163265
-97.1355621
300 W Oak St
Denton TX 76201
United States
```

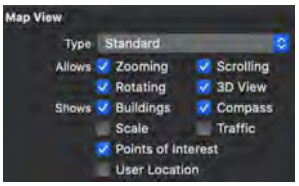


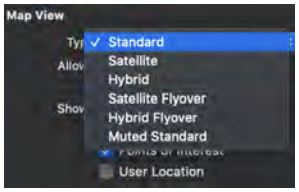
# Frameworks
# MapKit




Maps require MapKit framework

Add in IB or in code




Various types and settings including showing User Location
  Requires Location authorization from user

Frameworks
MapKit

Set region of map with outlet to MapView

```
let coord = CLLocationCoordinate2D.init(latitude: 32.0, longitude: -110.0)
let span = MKCoordinateSpan.init(latitudeDelta: 0.5, longitudeDelta: 0.5)
let region = MKCoordinateRegion.init(center: coord, span: span)
mapView.setRegion(region, animated: true)
```

```
mapView.userTrackingMode = .
    MKUserTrackingMode follow
    MKUserTrackingMode followWithHeading
    MKUserTrackingMode none
The map follows the user location.
```



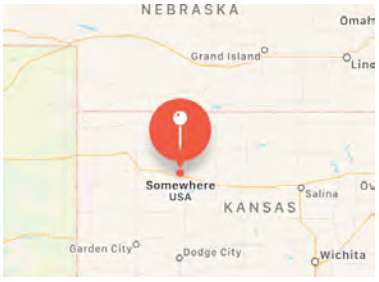
Frameworks
MapKit

Annotations

```
func addAnnotations() {
    let ann = MKPointAnnotation.init()
    ann.coordinate = CLLocationCoordinate2D.init(latitude: 39.0, longitude:
        -100.0)
    ann.title = "Somewhere"
    ann.subtitle = "USA"
    mapView.addAnnotation(ann)
    mapView.showAnnotations(mapView.annotations, animated: true)
}
```

Create coordinate
Set title and subtitle (optional)

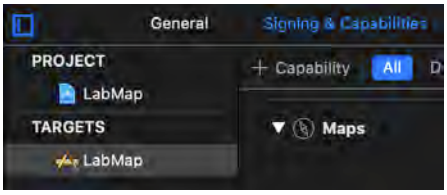
Add annotation to Map View




Can have Map View zoom to show set of annotations

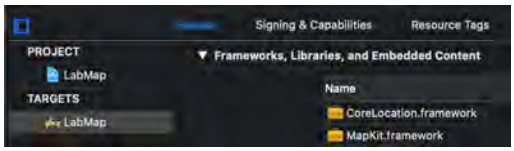
# Lab 21: Maps and Location

1. Create a new project called LabMap
2. In Main.storyboard, drop a MapView on the UI
3. Create and outlet to it in ViewController.swift named mapView




4. Signing & Capabilities: add Maps

5. General: Add CoreLocation framework




Continue…



# Lab 21: Maps and Location

6. In ViewController.swift import the MapKit framework
7. Claim to implement the CL manager delegate
8. Create a property for the location manager
9. In viewDidLoad…
   1. Check for location services. If enabled…
   2. Instantiate the manager
   3. Set the delegate
   4. Request authorization

```
import MapKit

class ViewController: UIViewController,
    CLLocationManagerDelegate {

    var locMgr : CLLocationManager?
    @IBOutlet weak var mapView: MKMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        if CLLocationManager.locationServicesEnabled() {
            locMgr = CLLocationManager()
            locMgr?.delegate = self
            locMgr?.requestWhenInUseAuthorization()
        }
```

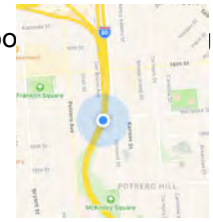
Continue…

# Lab 21: Maps and Location

```
func locationManager(_ manager: CLLocationManager,
                     didChangeAuthorization status: CLAuthorizationStatus) {
    if status == .authorizedWhenInUse {
        mapView.userTrackingMode = .follow
```

10. Implement the "didChangeAuthorization status" func to check the status and follow the user.

- Bonus: Set the ViewController to be the Map's delegate. Implement some functions from the related protocol.
- Bonus: Add an annotation to the map.
- Bonus: Add a long press gesture to the map. In the action func, get the view po press, convert that to a coordinate and drop an annotation.

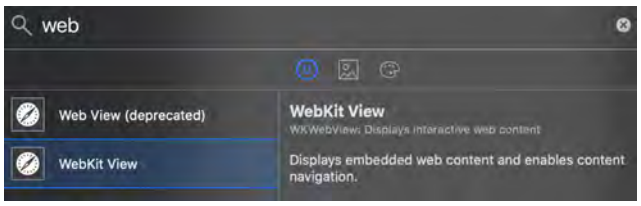


See Lab21.zip for help/solution and more.



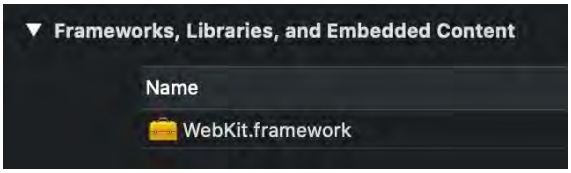
# WebKit
# WebView

IB or in code; import WebKit



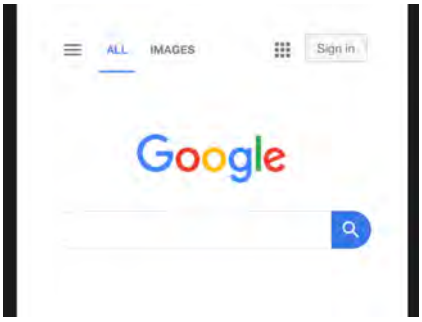

Loads: HTML, URL/URLRequest, PDF, media

Evaluates Javascript



WebKit
WebView

```
let url = URL(string: "https://www.google.com")
webView.load(URLRequest.init(url: url!))
```

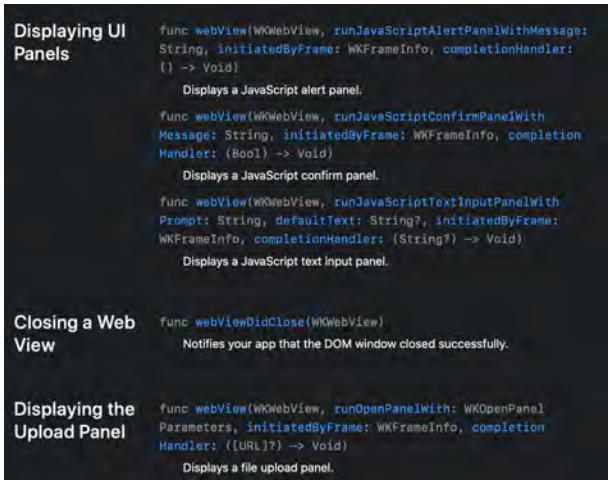





WebKit
WebView

WKUIDelegate

- funcs related to opening of windows, augment behavior of menu items and other UI tasks.



WebKit
WebView

WKNavigationDelegate - process callbacks (e.g., did finish) and authentication challenge

Decide policy for navigation action - should allow? cancel?




WebKit
WebView

```
func webView(_ webView: WKWebView,
             decidePolicyFor navigationAction: WKNavigationAction,
             decisionHandler: @escaping (WKNavigationActionPolicy) -> Void) {
    if navigationAction.request.url?.absoluteString.contains("google") == true {
        decisionHandler(.allow) // keep them in google
    }
    else {
        decisionHandler(.cancel)
    }
}
```

Call to determine if navigation is allowed:
If they stay in “google”, then it's allowed.

WebKit
WebView

```
func webView(_ webView: WKWebView, didReceive challenge: URLAuthenticationChallenge, completionHandler:
    @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {
    let user = "user"
    let pw = "pass"
    let credential = URLCredential(user: user, password: pw, persistence: URLCredential.Persistence.forSession)
    challenge.sender?.use(credential, for: challenge)
    completionHandler(URLSession.AuthChallengeDisposition.useCredential, credential)
}
```

Authentication Challenge

Create URLCredential with persistence setting
Use credential on challenge
Call completion handler



WebKit
WebView

backForwardList and go func (to item in list)

isLoading and estimatedProgress

canGoBack/Forward and goBack/Forward

reload and stopLoading

takeSnapshot (image)

allowsBackForwardNavigationGestures (swipes) property

## SafariServices

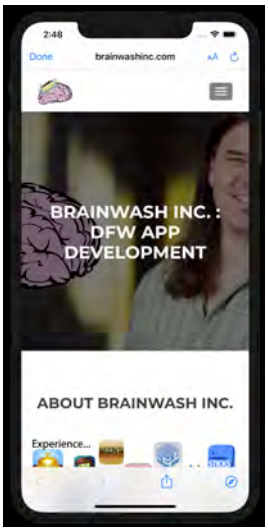
```
let sVC = SFSafariViewController(url: URL(string: "https://www.brainwashinc.com")!)
self.present(sVC, animated: true, completion: nil)
```

SafariViewController

Good for presenting a web page as a web page

Like Safari in your app: controls, buttons, bookmarks, etc.

Not much control - delegate with basic callbacks for activity controller items, loading and redirect






## Lab 22: WebView and JavaScript

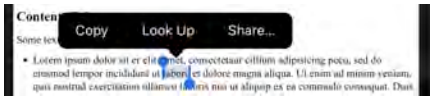
1. Open Lab21-Start.zip and open the project in Xcode
2. Review the index.html file
3. Run the app and verify it displays the contents of index.html in the web view
4. Review the project and notice WebKit has been added to the frameworks
5. Review ViewController and notice…
    1. Outlets to the UITextView and WKWebView
    2. The loadHtml function loads the index.html

Continue…

# Lab 22: WebView and JavaScript

Detecting when the user selects text in the web view:

6. Override the function named canPerformAction which takes 2 parameters and returns Bool.
    1. Print the action
    2. Return false
7. Run the app and long-press on some text.
8. Notice the function is called with a variety of actions - for each of the ones we return false for, they will not show up in the popup menu.





Now that we detect the selection, we want to have a function to call and run some JavaScript in the web view…

Continue…

```
override func canPerformAction(_ action: Selector, withSender sender: Any?) -> Bool {
    print (action)
    return false
}
```



# Lab 22: WebView and JavaScript

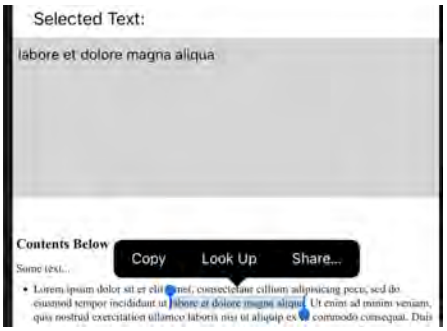
9. Create a function named webviewGetSelection that takes 2 parameters:
    1. webView: WKWebView
    2. completion: Any Error Void
10. Use Bundle.main.url(forResource: "getSelection", withExtension: “js") to get the URL to the JavaScript file
11. Load the contents of the file into a String
12. Use the evaluateJavaScript function to run the JavaScript
13. Call the completion handler with the parameters of the callback from step 9.

- Now we need to call this function when the selection is made…

Continue…

```
func webviewGetSelection(webView: WKWebView, completion : @escaping (Any?, Error?)->Void)
{
    guard let url = Bundle.main.url(forResource: "getSelection", withExtension: "js") else { return }
    guard let js = try? String.init(contentsOf: url) else { return }
    webView.evaluateJavaScript(js) { (result, error) in
        print (result ?? "no result")
        completion(result, error)
    }
}
```

# Lab 22: WebView and JavaScript




14. Back in canPerformAction, call webviewGetSelection
15. In the closure passed in, set the text selected by the user in the text view.
16. Run the app and verify the text selected in the web view appears in the text view.

```
override func canPerformAction(_ action: Selector, withSender sender: Any?) -> Bool {
    print (action)
    webviewGetSelection(webView: self.webView) { (result, error) in
        print (error ?? "no error")
        print (result ?? "no result")
        guard let r = result as? String else { return }
        self.tvText.text = r
    }
    return false
}
```



# Lab 23: More WebView and JavaScript

In this project we want to do several things…

- Define a web view in code for the UI
- Load HTML from a file in a web view
- Inject a user script into the web view creation for background color for each page
  - This is done by creating the web view with a configuration
- Set in the web view configuration to have a JS function created in the web view
- Handle the function call in the app code


1. Open Lab23-Start.zip and open the project in Xcode
2. Notice WebKit has been added to the frameworks.
3. Review index.html
4. In ViewController, notice…
    1. The WKWebView class property
    2. The loadHTML function's implementation
    3. The defineWebView function's implementation

Continue…

# Lab 23: More WebView and JavaScript

Initializing the web view with a configuration including a user script:

5. In defineWebView, add new code above the web view creation:
    1. Create an instance of WKWebViewConfiguration using the default initializer
    2. Create an instance of WKUserContentController using the default initializer

```
let wvConfig = WKWebViewConfiguration()
let wvContentController = WKUserContentController()
```

Continue…



# Lab 23: More WebView and JavaScript

3. Create a String with JavaScript to be injected into the web view:
   e.g., document.body.style.background = “#711";
4. Create an instance of WKUserScript with the string in step 3, atDocumentEnd and false as the 3 parameters.
5. On the WKUserContentController instance, call .addUserScript and pass in the instance created in step 4.
6. Set the WKUserContentController instance on the userContentController property on the WKWebViewConfiguration
7. Change the initialization of the web view to take the configuration

Continue…

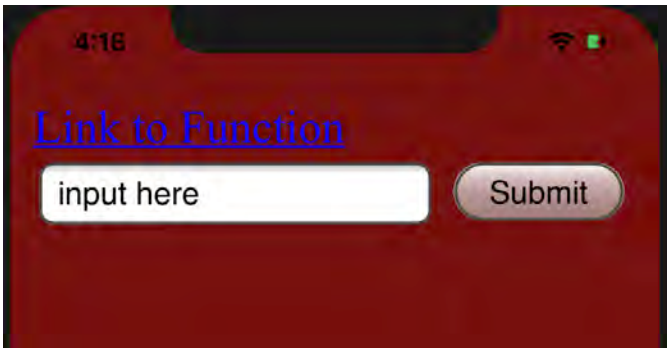
```
let source = """
    document.body.style.background = "#711";
"""
let userScript = WKUserScript(source: source, injectionTime: .atDocumentEnd, forMainFrameOnly: false)

wvContentController.addUserScript(userScript)
wvConfig.userContentController = wvContentController

webView = WKWebView(frame: self.view.bounds, configuration: wvConfig)
```

# Lab 23: More WebView and JavaScript

6. Run the app and verify the injected code succeeds

You may need to pinch to zoom in. In the Simulator, use the option key to mirror the cursor/finger.

Continue…






# Lab 23: More WebView and JavaScript

7. Before creating the web view, add a script message handler to the configuration.
   - This will create a function in the web view and specify what class has the function in the app to call when it's called in the JavaScript.

```
wvContentController.add(self, name: "handleMsg")
```

8. Declare to implement the WKScriptMessageHandler protocol (or create an extension of UIViewController to conform to the protocol)
9. Implement the userContentController function
10. Optionally bind the message.body value as a String
11. Print out the message body value
12. Run the app and verify the value in the input field is printed to the console in Xcode.

```
extension UIViewController: WKScriptMessageHandler {
    public func userContentController(_ userContentController: WKUserContentController, didReceive message: WKScriptMessage) {
        if let messageBody = message.body as? String {
            print(messageBody)
        }
    }
}
```

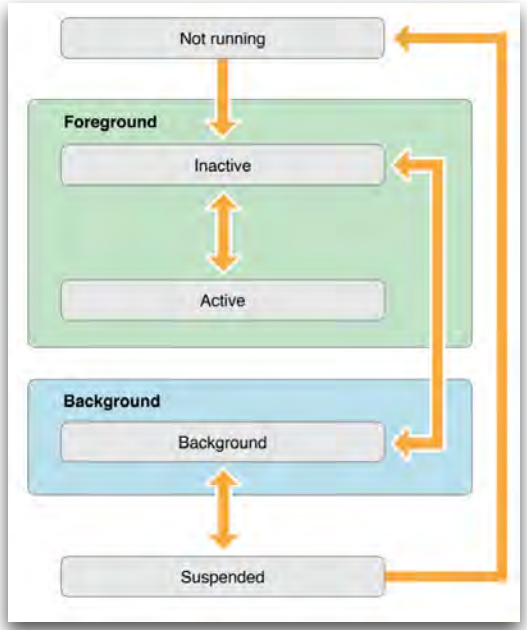
## App Dev Considerations
## App States

Not Running - Not launched yet or terminated
Inactive - In foreground but not receiving events. Brief. May be executing code.

Active - In foreground and receiving events. Normal foreground.

Background - In background and executing code. On way to being suspended (brief or request more time). Can be launched into background (notif, hot spot, etc.)

Suspended - In background but not executing code. In memory.




From https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html

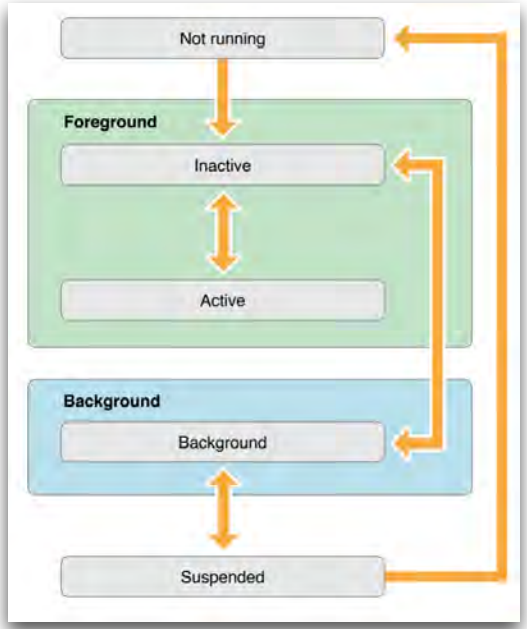


## App Dev Considerations
## App States

application:willFinishLaunchingWithOptions: - just launched

application:didFinishLaunchingWithOptions: - about to display

applicationDidBecomeActive: - coming to foreground




From https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html

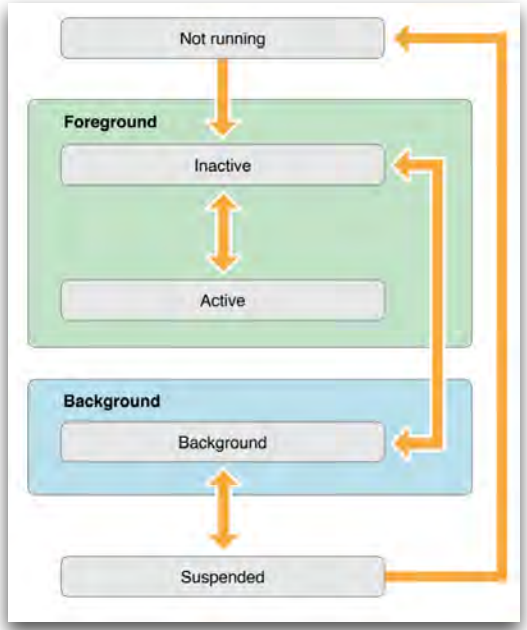
## App Dev Considerations
## App States

applicationWillResignActive: - leaving foreground, wrap up

applicationDidEnterBackground: - background, suspended soon

applicationWillEnterForeground: - back to foreground, not active

applicationWillTerminate: - about to terminate (not called if suspended)




From https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html



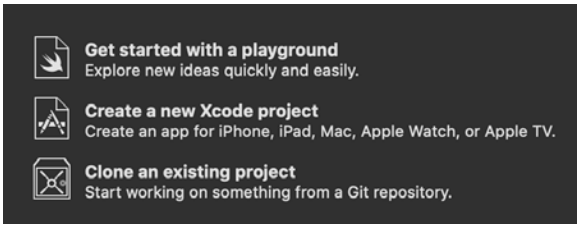
## App Dev Considerations
## App States

When moving away from active: store data, pause/cease UI updates, set timers/notifications

When moving back to active: load data, prep UI updates, cancel timers/notifs
Background modes are available, use them with consideration to power/battery, data, etc.

When the system needs memory, largest goes first.

## App Dev Considerations
## Git



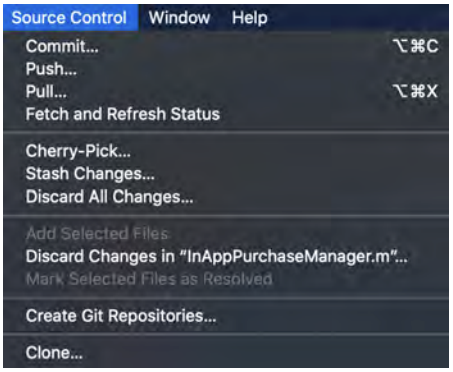

Xcode supports Git Source Control

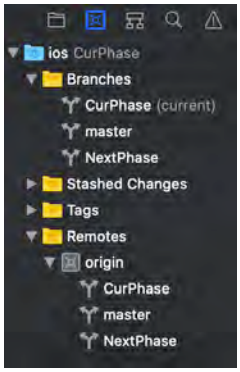
GitHub app or Xcode

Clone a project with URL



## App Dev Considerations
## Git

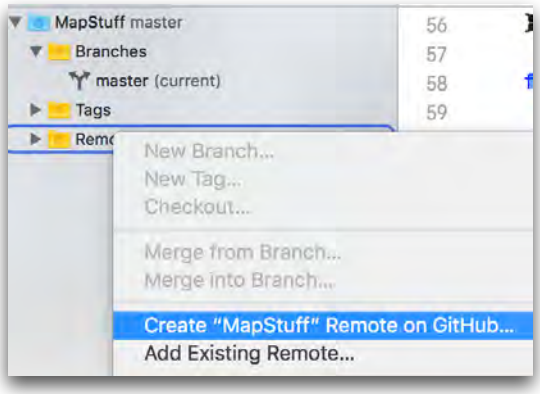



Changes, Additions are shown by files

Commit via Source Control menu

## App Dev Considerations
## Git




Repository created locally

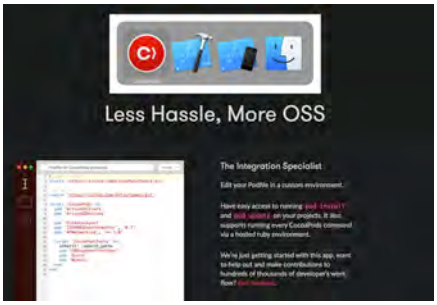
Create repo on Github from Remotes item



## App Dev Considerations
## CocoaPods

From: cocoapods.org

CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It has over 69 thousand libraries and is used in over 3 million apps. CocoaPods can help you scale your projects elegantly.



## App Dev Considerations
## CocoaPods

Install cocoaPods:

```
$ sudo gem install cocoapods
```

In project dir: pod init

Creates Pod.file
  Add pods you want installed

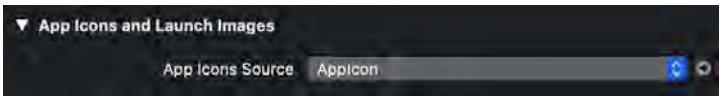
```
platform :ios, '8.0'
use_frameworks!

target 'MyApp' do
  pod 'AFNetworking', '~> 2.6'
  pod 'ORStackView', '~> 3.0'
  pod 'SwiftyJSON', '~> 2.3'
end
```

Run: pod install

Combines your project and Pods project into XCWorkspace



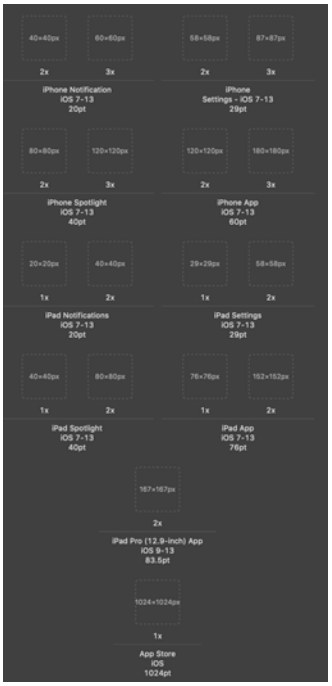
## App Dev Considerations
## Icons




In Assets catalog by default

Place for each size

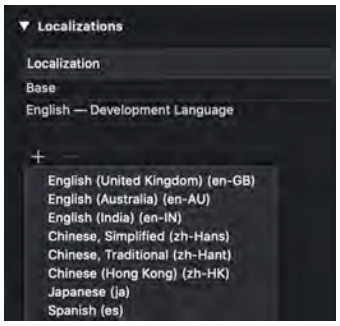
Converted to JSON

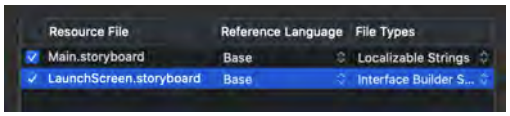
# Localization

Prepares your app for supporting multiple languages/translations

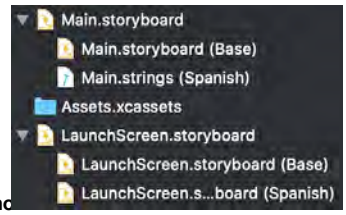
Project level - add languages




Select how to support in Storyboards: strings or storyboards




Files created accordingly



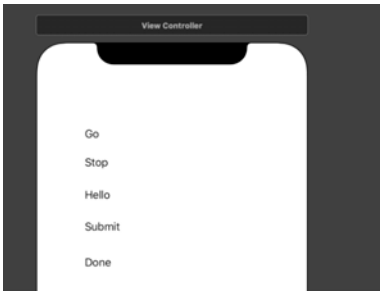


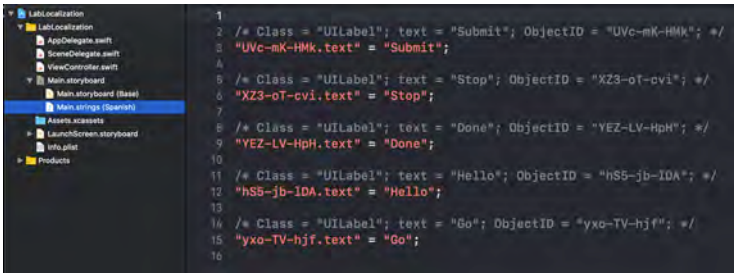

# Localization

New Storyboard for Launch

Strings file for Main

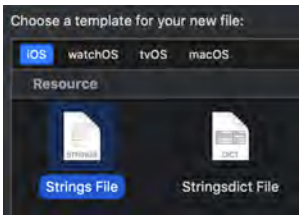
Get strings translated and that language will see those in the UI



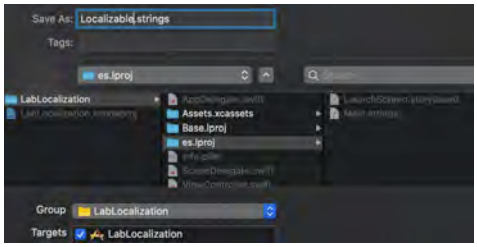
# Localization

For code you need a Localizable.strings file

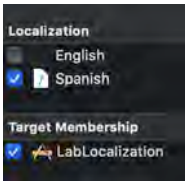



Add the file to the appropriate language directory

Verify it's associated with the right localization




Access in code

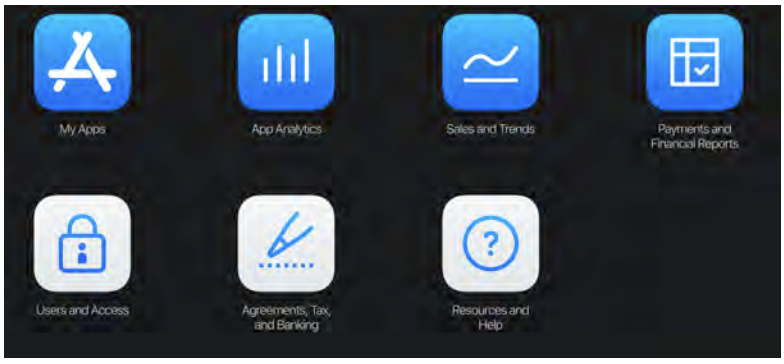



```
self.title = NSLocalizedString("Main", comment: "Main Title")
```



# Deployment

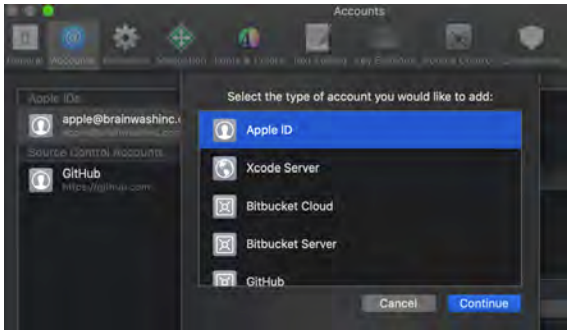
AppStoreConnect - https://appstoreconnect.apple.com/

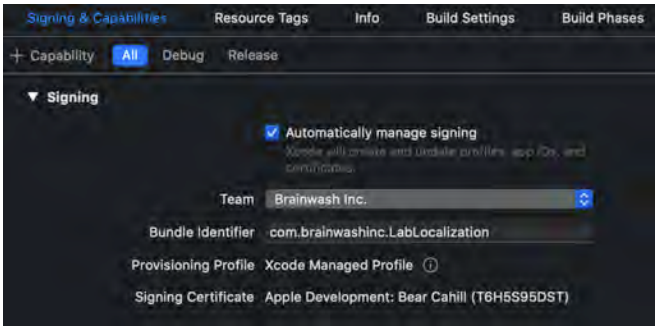


# Deployment

Add you Apple Developer account to Xcode

Have Xcode handle signing and certificates



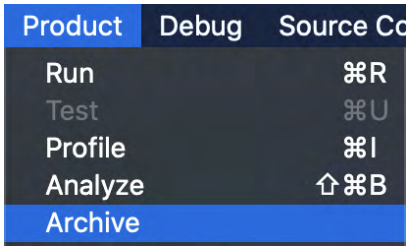


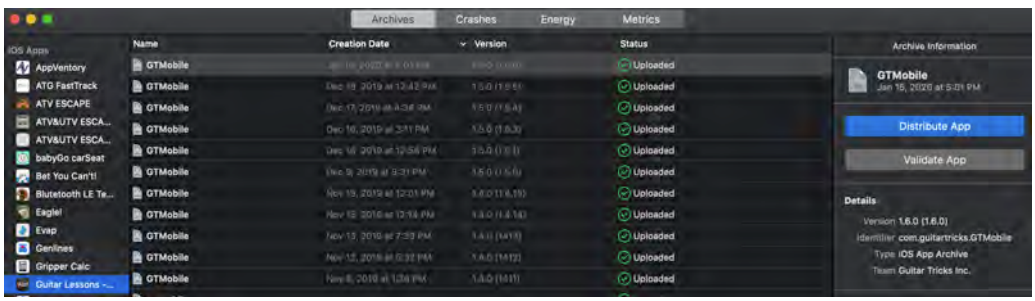

# Deployment

Build with Produce > Archive

Opens the Organizer

You can Distribute the app from here



## Deployment

TestFlight allows for internal (iTunes Connect account) and external (10k, no account needed)

Can invite via website or link

Users get TestFlight app and can install after invited/accepted



## The End