3.0 Lesson - Your First iPhone App

Paul Solt - Paul@SuperEasyApps.com
SuperEasyApps.com

3.1 Lecture - Your First iPhone App With Physics Based Animation

Watch how physics animations look on your iPhone app. In this lesson you will learn how to animate `UILabels` (i.e. text labels) using Swift code and spring physics.

Speed Coding Lesson Links

- 3.1 Lecture - Your First iPhone App With Physics Based Animation
- 3.12 Speed Coding - How to Add UILabels Programmatically in Swift
- 3.13 Speed Coding - How to Animate Your iPhone App Using Spring Physics

3.2 Tutorial - Create a New Xcode 7 Project Using Swift 2

1. Create Your First Xcode Project

- `File > New > Project`
    - `Command + Shift + N`
- `iOS > Single View Application > Next`
- **Product Name**: Any Descriptive App Name (Don’t use special characters)
- **Organization**: Your Name or Company Name
- **Organization Identifier**: A unique value (i.e. your website in reverse)
- **Language**: Swift
- **Devices**: Universal

- **Uncheck**: Core Data, Unit Tests, or UI Tests

2. Cleanup ViewController.swift

You can remove the `didReceiveMemoryWarning()` method in your ViewController.Swift file.

3. Run Your iPhone App

Press the Run button (Play button) to start your iPhone app. Or you can use the keyboard shortcut: `Command` + `R`

4. Change Your iOS Simulator

I recommend the iPhone 6S size iPhone simulator, change the starting simulator in the top left toolbar (blue icon). Depending on your Mac you might even want to use an iPhone 4S or iPhone 5S Simulator (especially if that's the same as the iPhone you own).

An iPhone 6S+ is a very large iPhone screen that will run slower because it has more pixels to simulate and it will be hard to see all the details when it's zoomed in/out. You'll also see aliasing issues (graphics artifacting) with thin lines, art, or small text.

5. Resize the iPhone Simulator

When you first start your iPhone app you'll see a huge white screen appear (especially if you're on a small non-retina Macbook or external display).

Resize the iOS Simulator so that you can see the entire screen using the menu or keyboard shortcuts.

- `iOS Simulator > Window > Scale > 50%`
- 100%: `Command` + `1`
- 75% Zoom: `Command` + `2`
- 50% Zoom: `Command` + `3`
- 33% Zoom: `Command` + `4`
- 25% Zoom: `Command` + `5`

3.3 Tutorial - iPhone App Flow 101

1. Xcode Project Settings

- Access project settings from the left panel, leftmost tab: Project Navigator: `Command` + `1`
- AppDelegate.swift is the first code file that starts in your app
- Main.Storyboard is the starting point for your app's user interface
  - The arrow is the first screen that will start (Checkbox: Is Initial View Controller)
  - The UI first screen is the ViewController, which is attached to the ViewController.swift code file

2. Quick Help

In your Swift or Objective-C code files you can `Option` + `Left-click` to get Quick Help on any code attribute. In Section 5 you'll learn how to use Dash for faster documentation.

3. Skim the UIViewController Reference

Read over the UIViewController reference file using Quick Help. Just skim the document so you get an idea of what code Apple has already provided to you for each app screen. Every method is something that you can use in your app when you need it, for now just be aware Apple has APIs that you can use to help make apps responsive.

4. Enable Line Numbers in Xcode

Xcode doesn't enable line numbers by default, enable them by going to the Xcode preferences.

```
Xcode > Preferences > Text Editing > Check: Line Numbers
```

5. Change Font Sizes or Themes

You can change the font size of your code to make it easier to read on a small screen, or you can change the colors in the Xcode preferences.

```
Xcode > Preferences > Fonts & Colors
```

If you want to change the font sizes do the following steps:

1. Duplicate (+) a theme
2. Name it: i.e. "Default 14"
3. Select all the colored lines under the `Source Editor` tab
4. Click the `T` font button
5. Set a new font size to 14 (Menlo Regular)
6. Repeat the process for the `Console` tab
7. Choose your new theme

6. Understand the App Flow

Let’s add some print statements to show you how lines of code get executed.

In `AppDelegate.swift`, add a `print("1st")` statement to `application(_:didFinishLaunchingWithOptions:)`:

```
func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
    // Override point for customization after application launch.
    print("1st") // add a print statement here
    return true
}
```

In your `ViewController.swift` file, add a `print("2nd")` statement to `viewDidLoad()`

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    print("2nd") // Look at the console output at the bottom
}
```

Run your iPhone app and along the bottom you should see the print statements in the Console output:

```
1st
2nd
```

7. Responsive Apps

iPhone apps or any tactile app that responds to touch need to be responsive. Throughout this course you will be learning the best coding practices that will make your apps feel at home on their respective platforms.

You expect apps to respond instantly and dislike when they hang or become unresponsive. By leveraging best practices described in this course, you will learn how to make your apps fun to use.

Why do laggy or slow apps frustrated you? Think about it, when you take a pen cap off a pen, it pops off instantly.

When you use an iPhone app you expect that same level of responsiveness, and when there is any delay it frustrates you because it doesn't behave the same way a real world object does. Lag, even very small amounts (100ms) will cause frustration because it is wasting your time and feels broken.

8. AppDelegate.swift

When you want to enable local or remote notifications in your app or save and load data when your app starts or stops, you'll use the AppDelegate.swift file and one of the delegate methods that Apple has provided. Read the UIApplication reference to learn more about the available methods you can implement in your app.

9. Assets.xcassets

The Asset Catalog is where you'll setup your apps icons and add images and graphics that you can display in your iPhone app.

10. ViewController.swift

Your very first screen that starts when you run your app will be your ViewController.swift

11. LaunchScreen.storyboard

This is the very first screen that appears in your app. For now you can ignore it, since it is only displayed for a very short time when your app first loads, and then it is skipped after your app is already loaded.

You'll only need to update this when you're ready to submit your app to the App Store.

3.4 Tutorial - Design the Button and Label UI

1. Add UI to Your Main.Storyboard File

1. Open your `Main.Storyboard` (not your `LaunchScreen.storyboard`)
2. Add a text label: `UILabel`
3. Add a button: `UIButton`
4. Double-click to edit the text for the label or button

2. Use the iOS Simulator Hardware Buttons

There are some important hotkeys to memorize with the iOS Simulator so that you can test your apps work in different orientations.

- Use the Simulator menu options:
  - `Simulator > Hardware`
- Rotate your iOS Simulator
  - **Rotate Left**: `Command` + `Left-arrow`
  - **Rotate Right**: `Command` + `Right-arrow`
- **Home button**: `Command` + `Shift` + `H`

3.5 Tutorial - How to Connect App UI to Code

You will be using the Assistant Editor to connect your UI to code files. In Xcode you can link each UI screen that you design to a different UIViewController code file.

1. Create UI Connections via IBOutlets and IBActions

Create an IBOutlet for your text label and an IBAction for your button when it is pressed.

1. Open the Assistant Editor (Venn Diagram Button in the top right)
   1. **Assistant Editor** (2+ editors): `Command` + `Alt` + `Enter`
   2. **Standard Editor** (single editor): `Command` + `Enter`
2. Right-click and drag from the UIButton or UILabel (i.e. any UIView subclass)
   1. Trackpad tip: Two-finger click on the UI element and drag. Keep one finger pressed and lift and drag using your second finger.

2. Create an Action for a UIButton

Drag an action connection from the *Press Me* `UIButton` to the ViewController.swift code file in the Assistant Editor. Type the name *buttonPressed* and make sure you choose Action not Outlet from the pop-up—Xcode will auto-insert the code:

```
@IBAction func buttonPressed(sender: AnyObject) {

}
```

3. Create an Outlet for a UILabel

Drag an outlet connection from the *Hello!* `UILabel` to the ViewController.swift code file in the Assistant Editor. Type the name *helloLabel* and Xcode will auto-insert the code:

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

4. Make the Button Do Something

When you press the button there is no indication that anything is happening. Fix it by adding a print statement inside the curly braces of the `buttonPressed(_:)` method.

```
@IBAction func buttonPressed(sender: AnyObject) {
    print("button Pressed!")
}
```

5. Troubleshooting Issues

The Jump Bar

Along the top of Xcode's editor is the jump bar (i.e. bread crumbs). Use the jump bar to switch between code files or when your using the Assistant Editor to choose the Automatic option.

Manual vs. Automatic

In Xcode 7.2 there are some quirks with the Assistant Editor, you may need to switch your right-hand side from Manual to Automatic when you start Xcode (or when you open existing Xcode projects).

No File Selected Bug

If you are working on a project and you don't see anything, close Xcode and re-open it again. There are two ways to quit Xcode, you can use the menu bar or the keyboard shortcut.

- `Xcode > Quit`
- **Quit**: `Command` + `Q`

There are some open bugs and glitches with Xcode 7.2, if you see anything strange you can report them to bugreport.apple.com

6. Bug Fix Videos

If you encounter any issues, watch the Bug Fix videos to solve the most common problems for UI and Swift code.

- 3.10 Bug Fix - How to Fix 3 Common Crashes in Xcode 7
- 3.11 Bug Fix - How to Fix 3 Common Swift Code Errors

3.6 Tutorial - What Is Auto Layout and How to Add Auto Layout to Your UI

Auto Layout allows you to specify both position and size of every UI element in your iPhone app. In order to see an image, button, or label in your iPhone app you need to tell Xcode how big it is, and then you need to tell it where to be on the screen.

Auto Layout enables responsive layouts for different sized iPhone and iPad devices.

1. Notebook

Make sure you write down anything that seems strange or new to you. If you have any questions write them down. As you explore and ask questions you'll uncover answers to your questions.

1. What does "String?" mean?
2. what is func?
3. etc.

2. Programmatically Change a UILabel (Without Auto Layout)

In your `buttonPressed(_:)` method change the font using the `text` property.

```
@IBAction func buttonPressed(sender: AnyObject) {
    print("button Pressed!")
    helloLabel.text = "Coffee Time!"
}
```

Without Auto Layout, the text "Coffee Time!" is clipped because your label doesn't resize itself to show the new text.

3. Add Auto Layout to Your Storyboard UI

You will add Auto Layout constraints in the same way that you make outlet and action connections. The only difference is that you will drag from a UI element to another UI element.

1. Add the following constraints to your “Press Me” button
    1. *Vertical Spacing to Top Layout Guide*
    2. *Center Horizontally in Container*
2. Add the following constraints to your “Hello!” label
    1. *Vertical Spacing*
    2. *Center Horizontally*

Tip: Hold shift to select multiple constraints and then press the *Add Constraints* button.

4. Assistant Editor Preview

The Assistant Editor has a Preview option which will be very useful when you are designing the UI for your iPhone app. You can preview how it will look on different screen sizes (iPhone + iPad) and orientations (portrait vs. landscape).

1. Choose the Preview option from the Jump Bar when you have your `Main.storyboard` file open.
2. Select your ViewController to see how it will look
3. Add previews of different device sizes use the + button
4. Remove previews by selecting them and pressing the delete key

3.7 Tutorial - How to Programmatically Add a UILabel Using Swift 2

1. Declare two UILabel objects inside the ViewController.swift class

```
class ViewController: UIViewController {
    @IBOutlet weak var helloLabel: UILabel!
    // Declare labels here
    var welcomeLabel: UILabel!
    var nameLabel: UILabel!
```

2. Make a new method (chunk of code) called `addLabels()`

```
func addLabels() {
    welcomeLabel = UILabel()
    welcomeLabel.text = "Welcome!"
    welcomeLabel.font = UIFont.systemFontOfSize(36)
    welcomeLabel.sizeToFit()
    welcomeLabel.center = CGPoint(x: 100, y: 240) // 40)
    view.addSubview(welcomeLabel)
}
```

3. Call the `addLabels()` method in your `viewDidLoad()` method

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

    print("2nd")

    // Create the labels
    addLabels()
}
```

4. Press the Run button to start the app to see the new label
5. Create and customize the `nameLabel`

```
func addLabels() {
    welcomeLabel = UILabel()
    welcomeLabel.text = "Welcome!"
    welcomeLabel.font = UIFont.systemFontOfSize(36)
    welcomeLabel.sizeToFit()
    welcomeLabel.center = CGPoint(x: 100, y: 240) // 40)
    view.addSubview(welcomeLabel)

    nameLabel = UILabel()
    nameLabel.text = "Paul Solt"
    nameLabel.font = UIFont.boldSystemFontOfSize(48)
    nameLabel.sizeToFit()
    nameLabel.center = CGPoint(x: 200, y: 290) // 90)
    view.addSubview(nameLabel)
}
```

work! You've added a method that created two text labels programmatically. Now see if you can change some of the attributes for both labels (i.e. use your name, not mine!).

3.8 Tutorial - Add Animation to Your UILabel Programmatically

You are going to animate the `welcomeLabel` and `nameLabel`.

1. Add a new method called animateLabels()

```
func animateLabels() {
}
```

2. Call the method in your `buttonPressed(_:)` method, which will allow you to start the animation.

```
@IBAction func buttonPressed(sender: AnyObject) {
    print("button pressed!")

    animateLabels()
}
```

3. Add animation code using the `UIView` animation method

```
func animateLabels() {
    welcomeLabel.center = CGPoint(x: 100, y: 40)
    welcomeLabel.alpha = 0

    UIView.animateWithDuration(0.5, delay: 0.0, options: [], animations:
{

        self.welcomeLabel.center = CGPoint(x: 100, y:40 + 200)
        self.welcomeLabel.alpha = 1
        }, completion: nil)
}
```

4. Animate the `nameLabel`

```
func animateLabels() {

    welcomeLabel.center = CGPoint(x: 100, y: 40)
    welcomeLabel.alpha = 0

    UIView.animateWithDuration(0.5, delay: 0.0, options: [], animations:
{

        self.welcomeLabel.center = CGPoint(x: 100, y:40 + 200)
        self.welcomeLabel.alpha = 1
        }, completion: nil)

    nameLabel.center = CGPoint(x: 200, y: 90)
    nameLabel.alpha = 0 // 0 means invisible (range [0.0, 1.0])

    UIView.animateWithDuration(0.5, delay: 0.5, options: [], animations:
{

        self.nameLabel.center = CGPoint(x: 200, y: 90 + 200)
        self.nameLabel.alpha = 1 // 1 means visible
        }, completion: nil)
}
```

3.9 Tutorial - Make It Physical With Spring Physics Animations

Change the animations from linear animations to physics-based animations using the `animateWithDuration(_:delay:usingSpringWithDamping:initialSpringVelocity:options:animations:completion:)` method.

teest 123
1. Create a new `animateLabelsWithPhysics()` method

```
@IBAction func buttonPressed(sender: AnyObject) {
    print("button pressed!")

    //animateLabels()
    animateLabelsWithPhysics()
}
```

1. Call the new `animateLabelsWithPhysics()` method and comment out the `animateLabels()` method

```
@IBAction func buttonPressed(sender: AnyObject) {
    print("button pressed!")

    //animateLabels()
    animateLabelsWithPhysics()
}
```

3.10 Bug Fix - How to Fix 3 Common Crashes in Xcode 7

As you work with your first iPhone app you will inevitably run into different crashes. The three most common crashes are what every beginner encounters. If you know what to look for, you can avoid these in the future.

Two of the crashes are real crashes and one is because you clicked and mistakenly added a breakpoint, which is used for finding bugs.

Here are the three crashes and how to fix them:

1. **Breakpoint Crash**
   - Problem: `Thread 1: breakpoint 1.1`
   - Why: You clicked in the gutter left of your code and added a blue flag.
   - Solution: Delete or disable the breakpoint (blue flag) from the code gutter.
2. **Storyboard IBOutlet Crash**
   - Problem: `Thread 1: signal SIGABRT`

  - Console: `Terminating app due to uncaught exception 'NSUnknownKeyException', reason: '[<ViewController 0x7fbd74a090d0> setValue:forUndefinedKey:]: this class is not key value coding-compliant for the key spellingMistakee.'`
  - Why: You deleted an Outlet/Action connection in code or renamed it (i.e. spelling mistake).
  - Solution: Right-click on the UI element and remove the old Outlet/Action connection mentioned as the `key` in the top of the Console output.
3. **Nil Optional Crash**
   - Problem: `Thread 1: EXC_BAD_INSTRUCTION`
   - Console: `fatal error: unexpectedly found nil while unwrapping an Optional value`
   - Why: You removed a UIView IBOutlet connection and then used the IBOutlet variable in code. The variable has no value until you reconnect it to the UI element.
   - Solution: Right-click and drag from the UI element to the IBOutlet property that Xcode stopped at with the `EXC_BAD_INSTRUCTION`.

3.11 Bug Fix - How to Fix 3 Common Swift Code Errors

Code is case sensitive and a very strict speller. Computer code only works if you spell everything the same. Here are three common code errors you will encounter in Swift:

1. **Swift Compiler Error: Unresolved Identifier**
   - Problem: `Use of unresolved identifier 'welcomeLLabel'` (i.e. a spelling mistake!)
   - Why: You made a mistake typing a variable name. Swift is case-sensitive for all variable names, methods, and keywords.
   - Solution: Fix any typos and make sure your variables or methods match exactly. Unresolved identifier is a clue that you didn't declare a variable near the top of your app, or you misspelled the variable when trying to use again.
2. **Swift Compiler Error: Missing Matching Parenthesis ( )**
   - Problem: `Use of local variable ‘addLabels’ before its declaration` (misleading clue)
   - Solution: Make sure every left parenthesis (open) has a matching right parentheses (close). Method and function calls should look like: `addLabels()`
3. **Swift Compiler Error: Missing Matching Curly Braces { }**
   - Problem 1: `Expected declaration` (one of many errors, at end of file)
   - Problem 2: `Only instance methods can be declared @IBAction` (misleading clue,

look above error)
  - Solution 2: Add a missing curly brace to a method, function, closure, or class declaration.

3.12 Speed Coding - How to Add UILabels Programmatically in Swift

Add text labels using both Storyboard files and programmatically using Swift code.

3.13 Speed Coding - How to Animate Your iPhone App Using Spring Physics

Add physics based animations to any view: labels, images, and other content in your iPhone app.

3.14 Tutorial - Install Your App on Your iPhone - No Apple Developer Program Required

To run your app on your iPhone you need a lightning cable, iPhone, and an Apple ID.

Open Xcode's preferences:
`Xcode > Preferences > Accounts > + >`

3.15 Code Exercise 1 - Hide the Labels on App Launch

3.16 Solution 1 - Hide the Labels on App Launch

3.17 Code Exercise 2 - Change the Welcome Name Using a UITextField

3.18 Solution 2 - Change the Welcome Name Using a UITextField

3.19 Quiz - Your First iPhone App (10)

Take the quiz and test your first iPhone app knowledge. You will need to use Apple’s documentation and other resources to answer the questions.

- 3.19 Quiz - Your First iPhone App