

with *Swift*

# What is Swift?

---

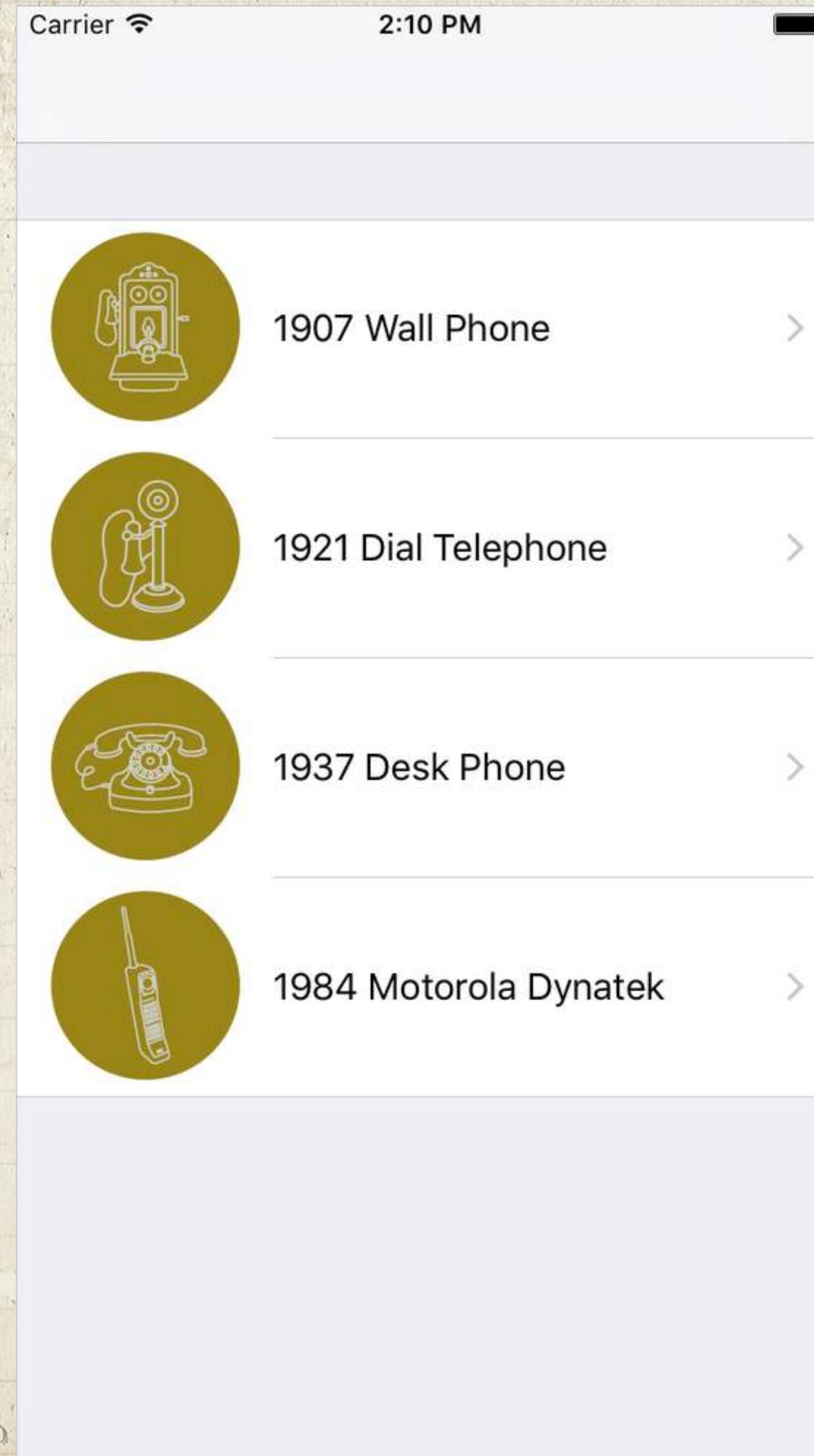
Swift is a new programming language introduced by Apple in June 2014 for creating iOS and OS X apps.



## Release History

First beta	June 2014
Swift 1.0	September 2014
Swift 1.1	October 2014
Swift 1.2	April 2015
Swift 2.0 beta	June 2015
Swift 2.0	September 2015
Swift 2.1	October 2015
Swift 2.2	March 2016
Swift 3.0	September 2016

# The App We're Going to Make



Good as Old Phones returns the phones of yesteryear back to their original glory and then gets them into the hands\* of those who appreciate them most.

Whether you're looking for a turn-of-the-century wall set or a Zack Morris special, we've got you covered. Give us a ring, and we'll get you connected.

\*Hands-free phones available

## CONTACT



phonestore@example.com

# You Need Xcode to Make iOS Apps

As we make the different parts of the app, we're going to show you short screencasts of us working in Xcode. Feel free to follow along with your own copy of Xcode.



APP Store



Xcode

# Level 1

## Xcode and Storyboards

### Section 1 - Creating an Xcode project



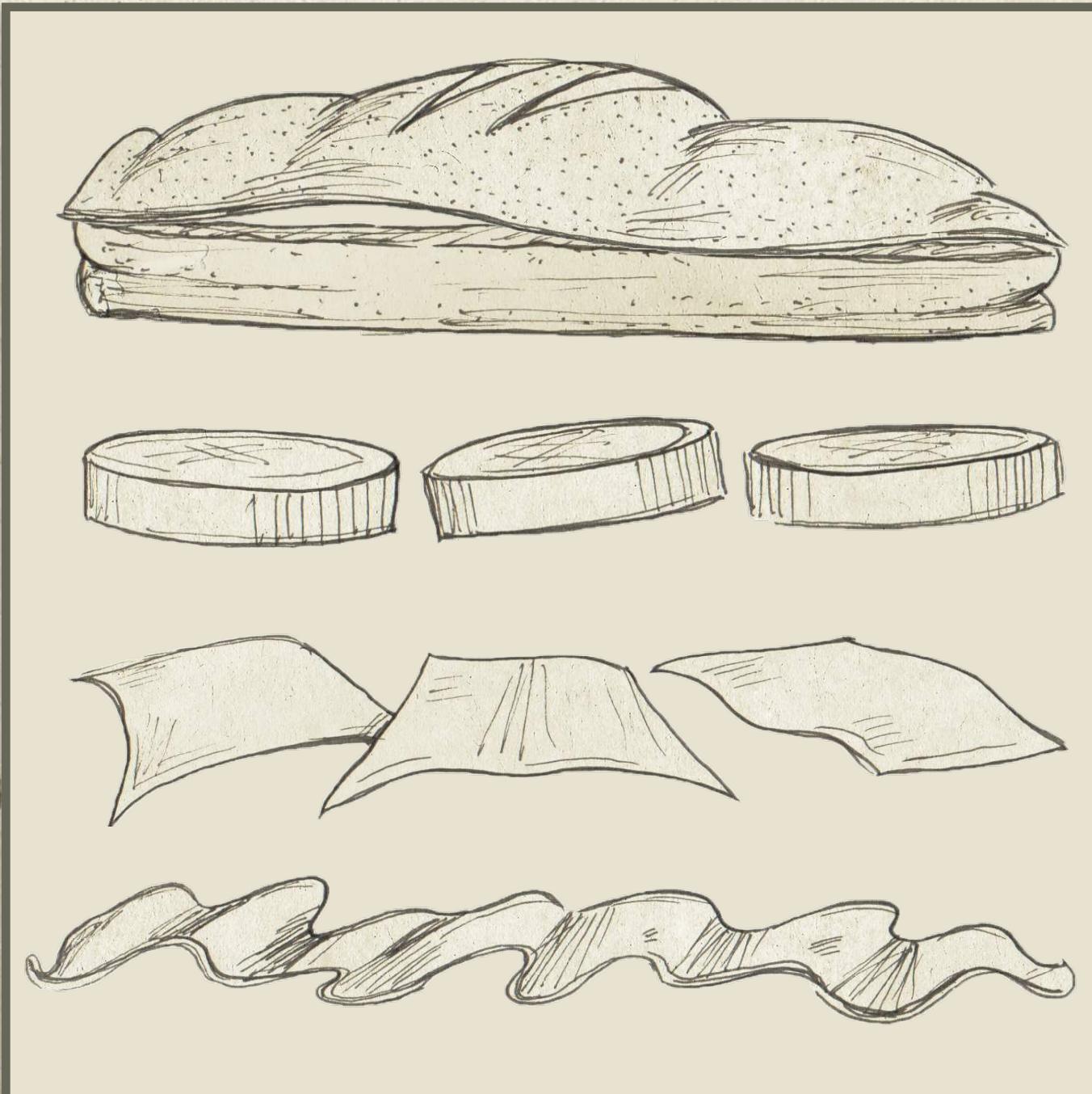
# Apps and Sandwiches

---

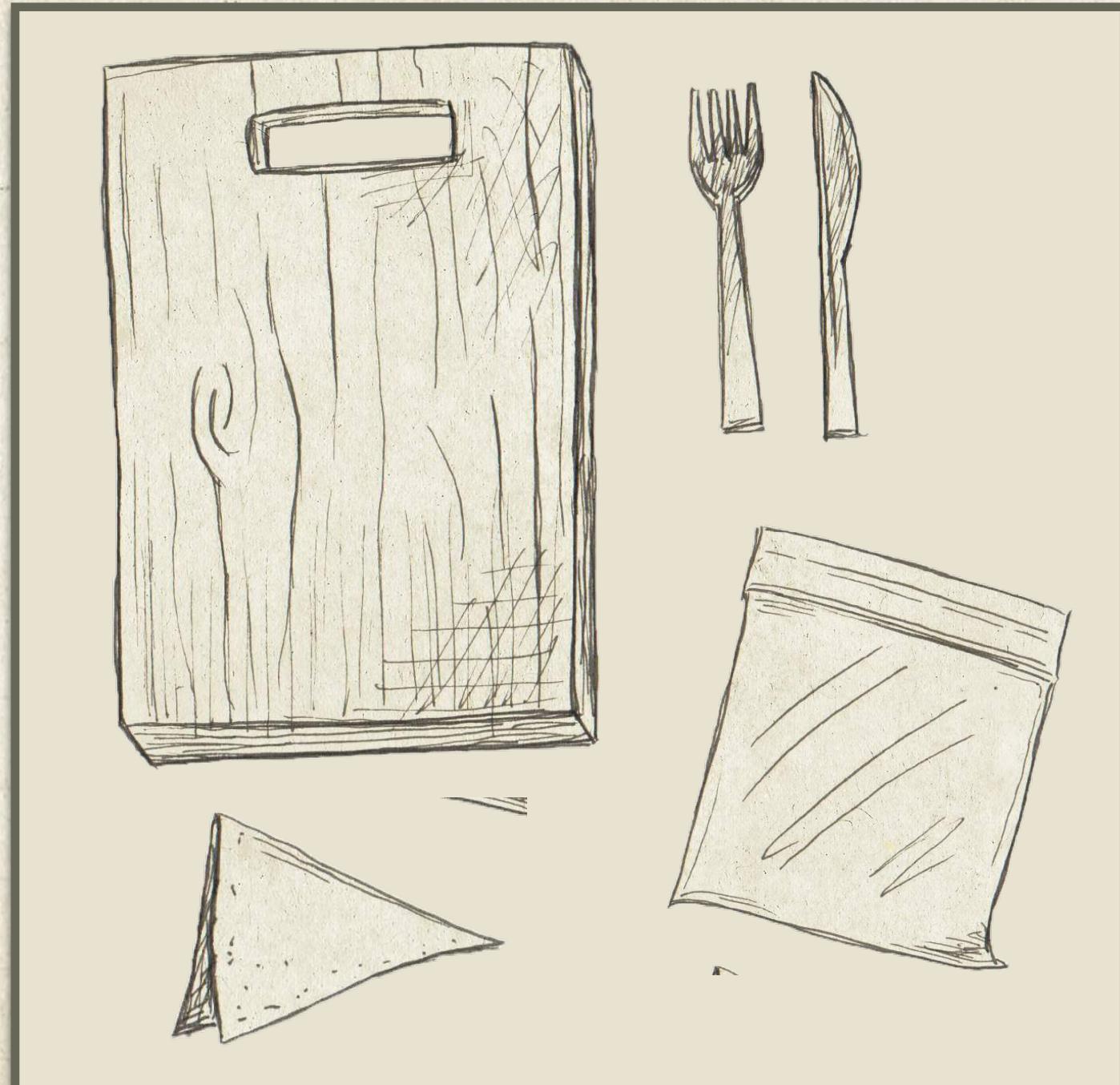


what do we need to  
make this sandwich?

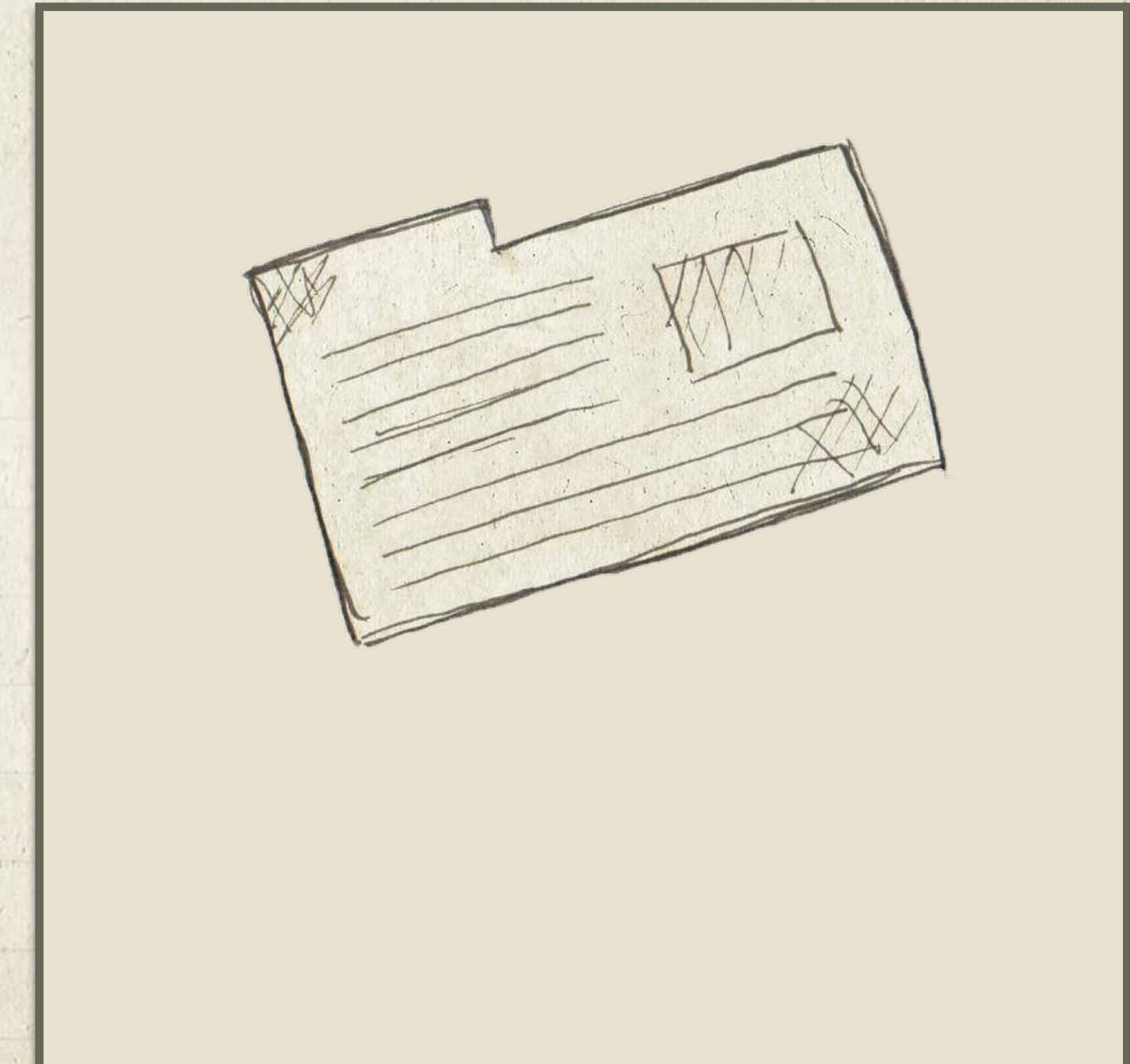
## Raw Ingredients



## Cooking Utensils



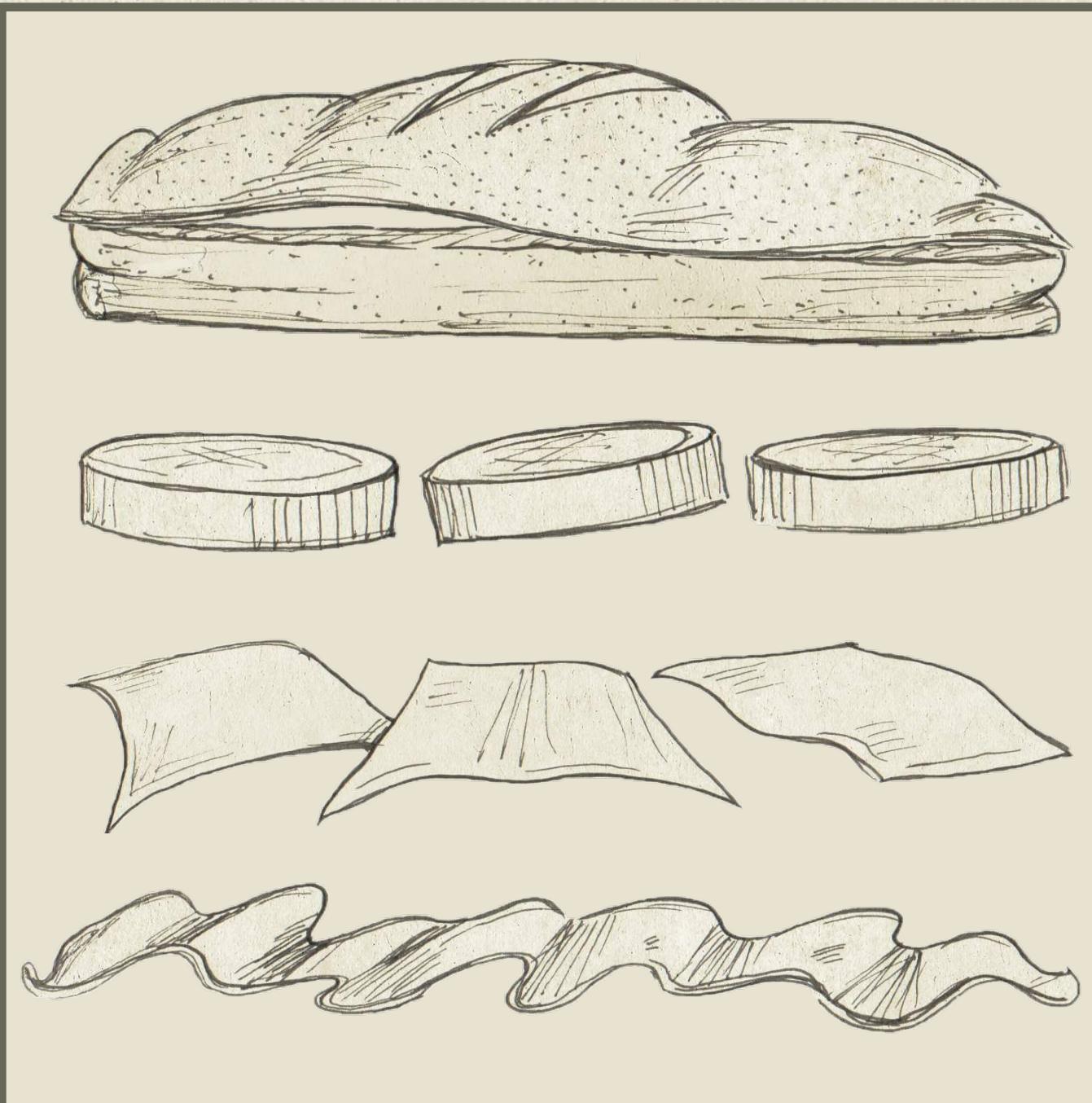
## The Recipe



# UIKit is Like the Sandwich Ingredients

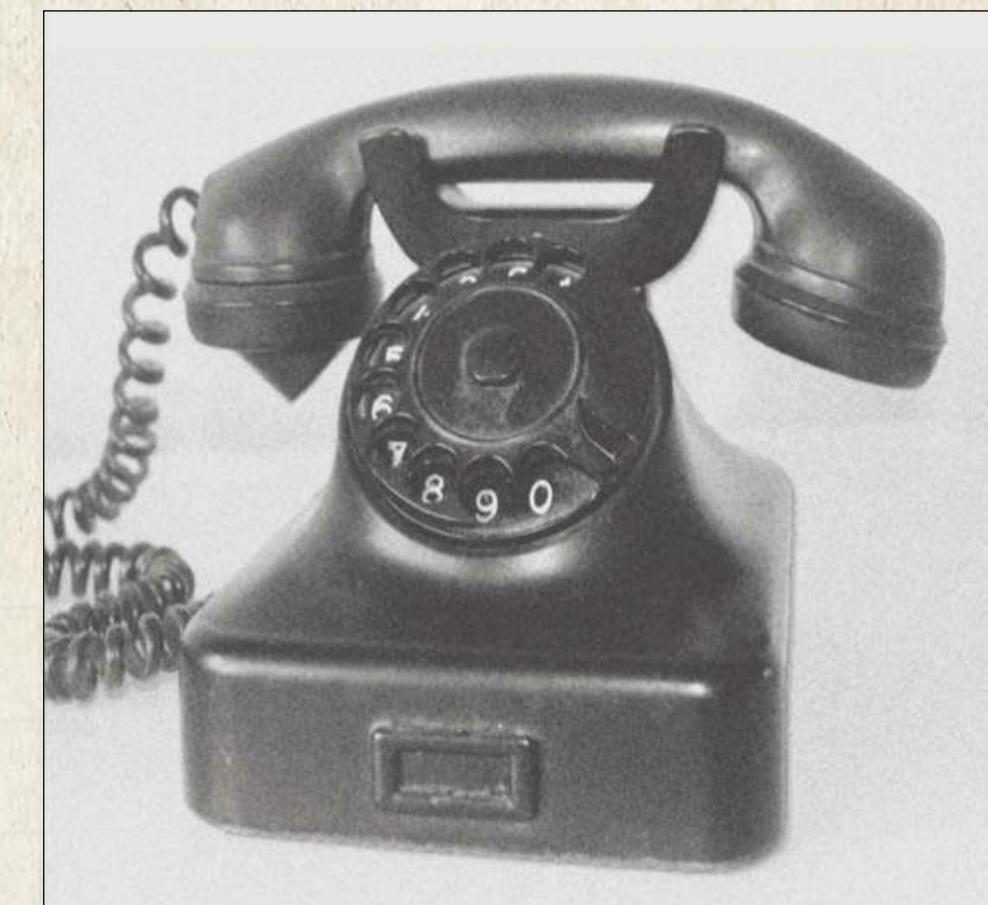
UIKit is a framework that contains templates for standardized ways to display data in your app

## Raw Ingredients



## Text

**1937 Desk Phone**



## Images

## Buttons

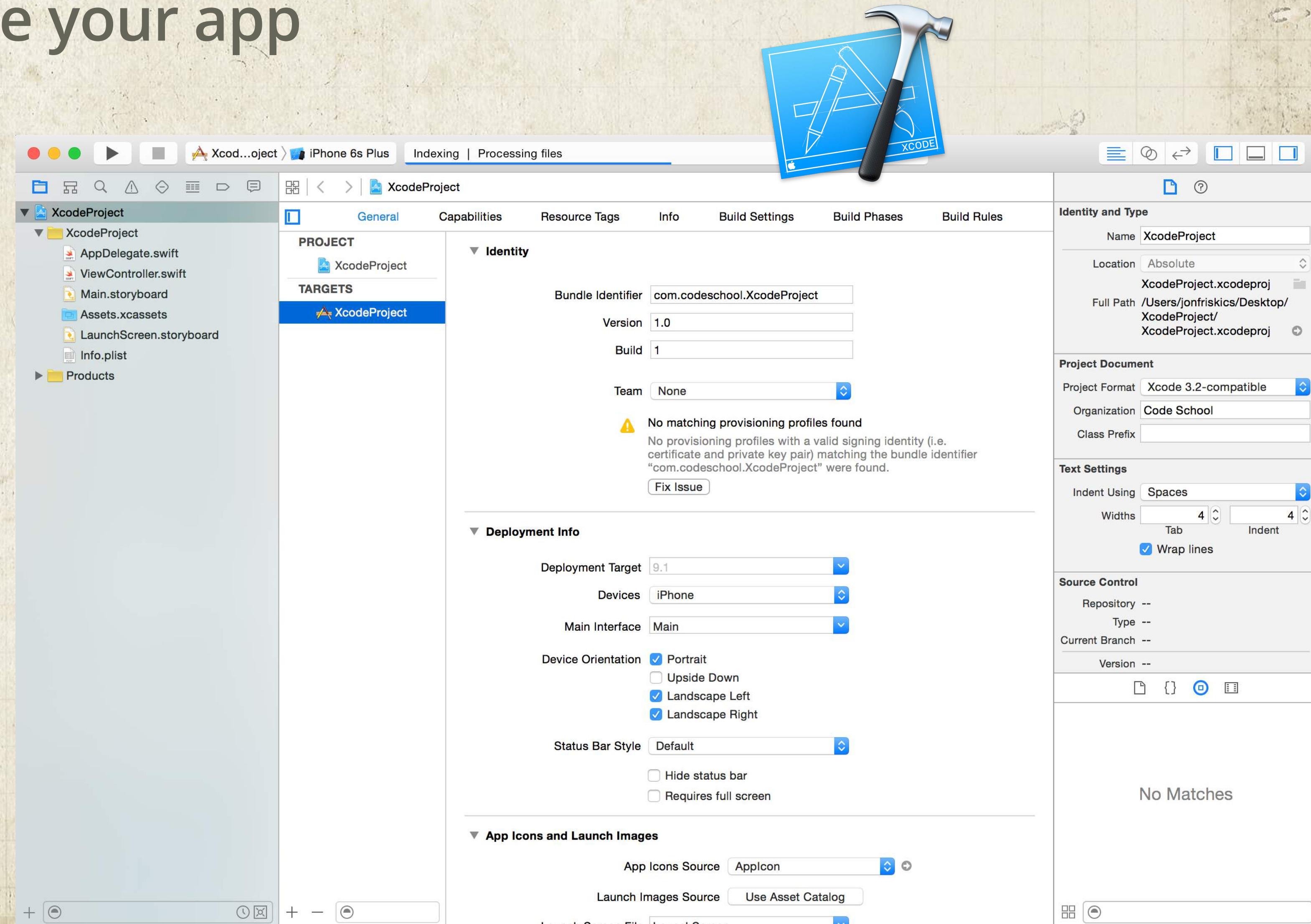
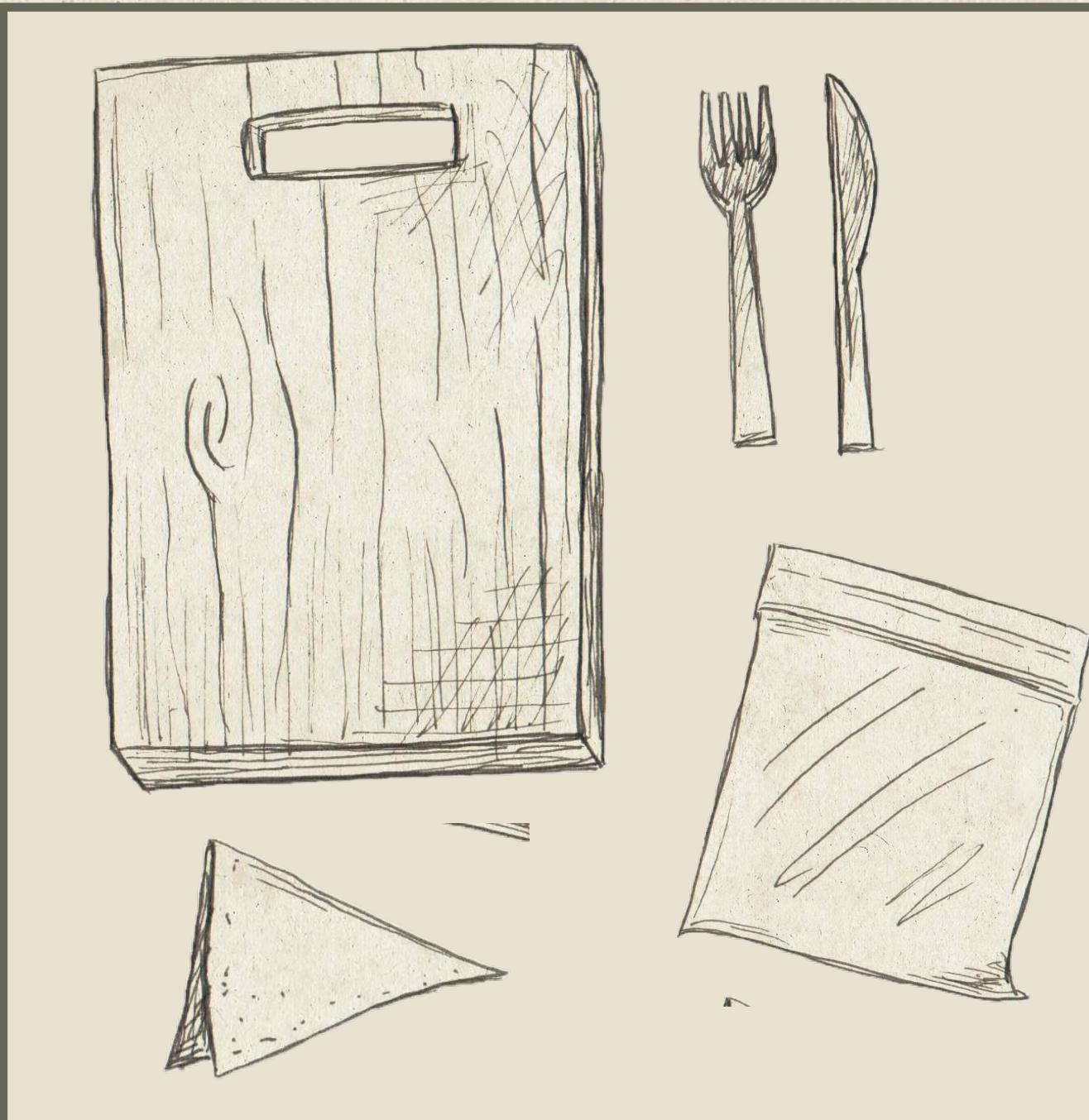
**ADD TO CART**

**APP  
EVOLUTION**  
*with Swift*

# Xcode is Like the Kitchen Utensils

Xcode is a free program provided by Apple that you use to write Swift code and assemble your app

## Cooking Utensils

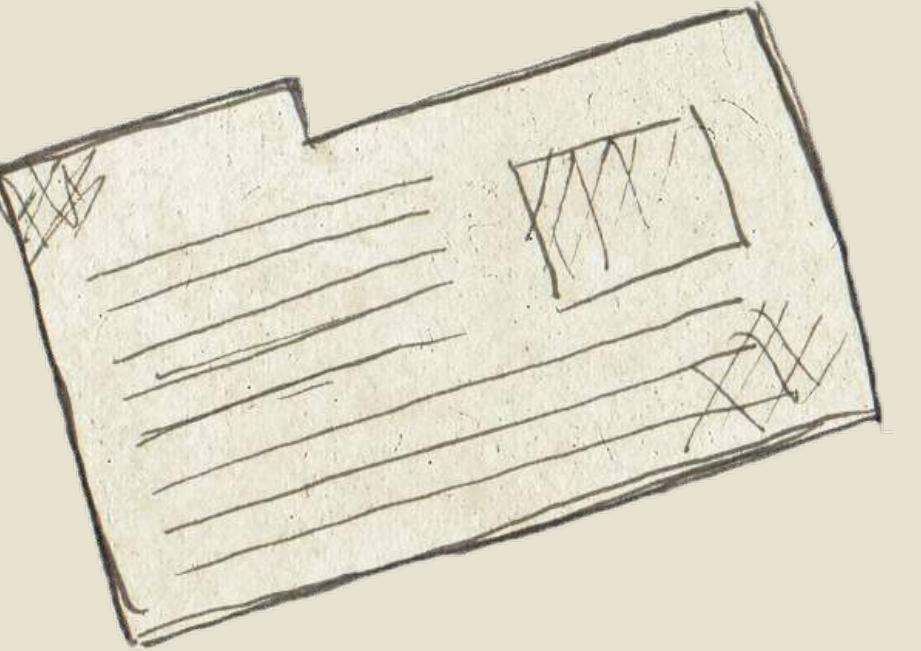


# Swift Is Like the Instructions for Making the Sandwich

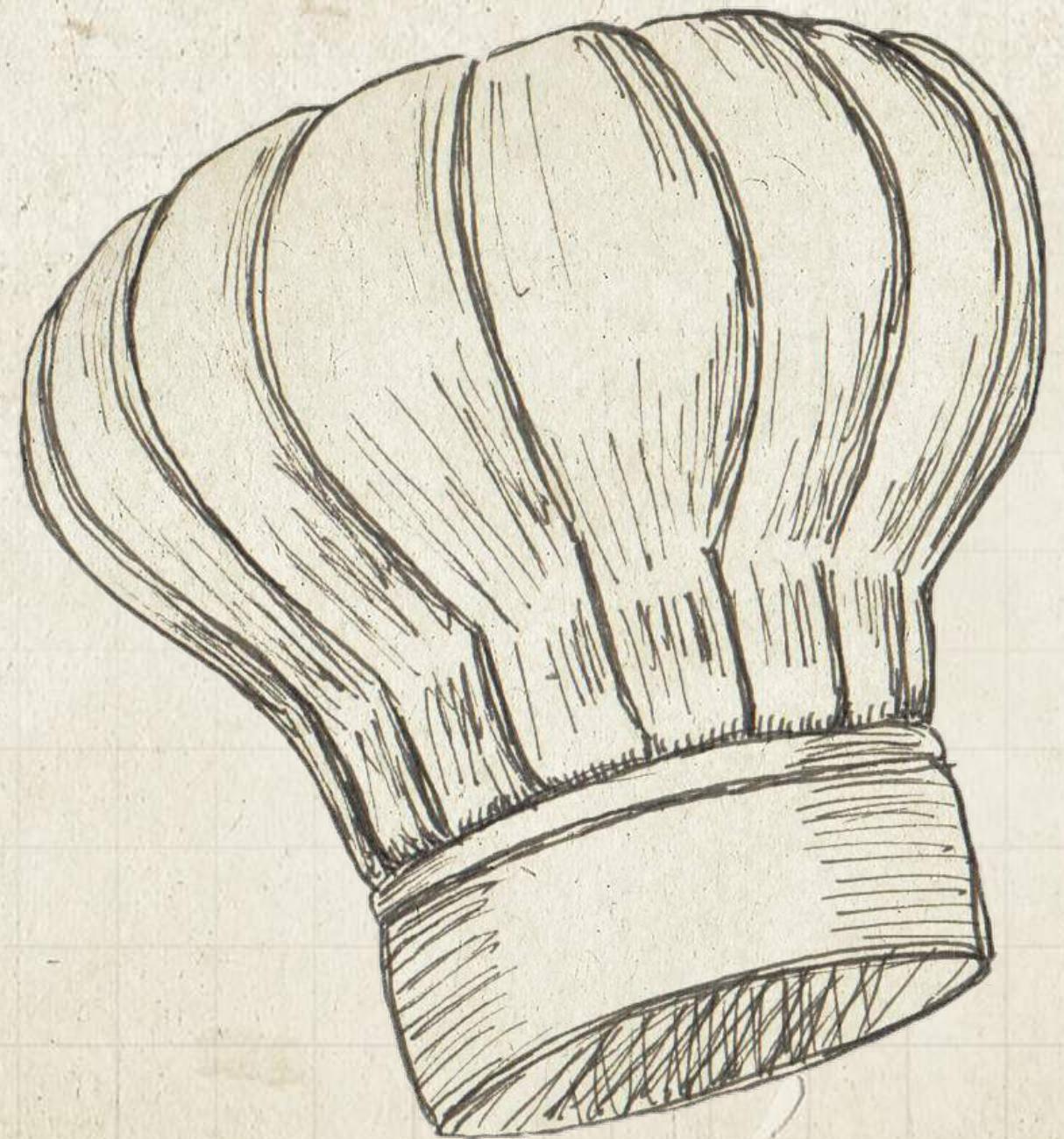
---

You'll write Swift code, in Xcode, that tells parts of the UIKit framework how to display your data.

The Recipe



You are the chef!



APP  
EVOLUTION  
*with Swift*

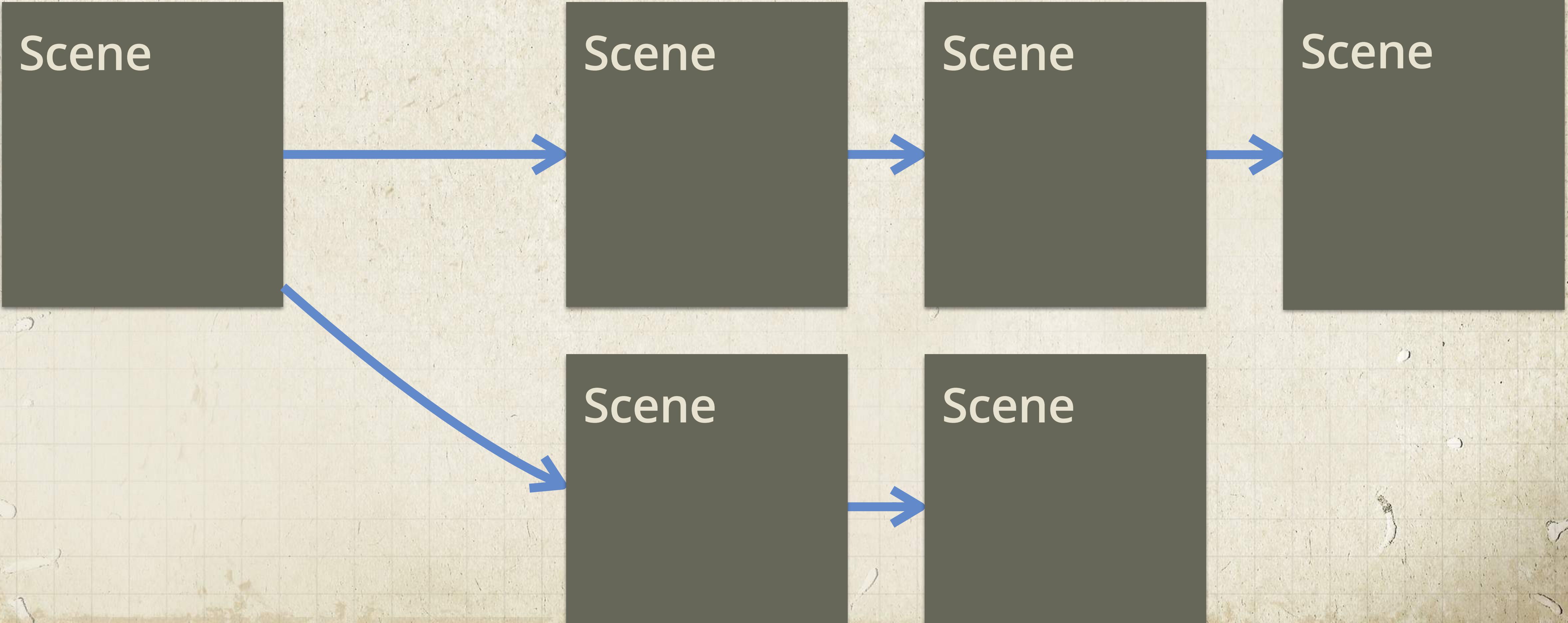
# Screencast: Create and Set Up a New Project

---



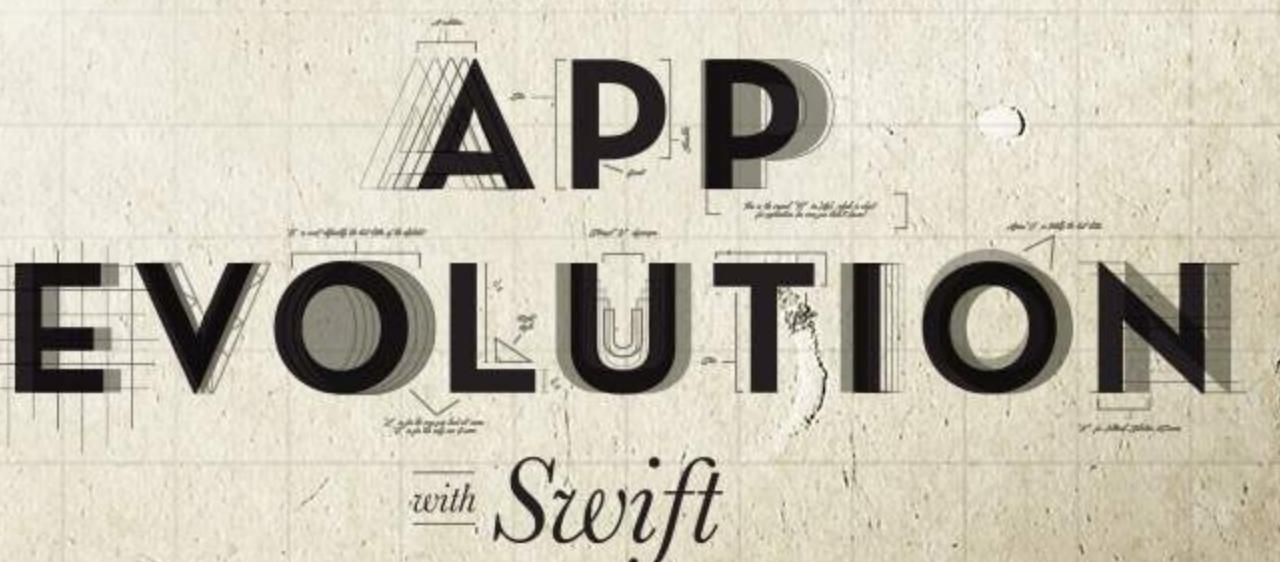
# Storyboards Let You Visualize Your App's Flow

A storyboard is a tool for laying out your UI and connecting the flow between different screens that your app will display.



# Each Scene Displays at Least 1 View

A single main view contains everything the user will see on that screen.



# Screencast: Adjusting the Color of the Main View

---



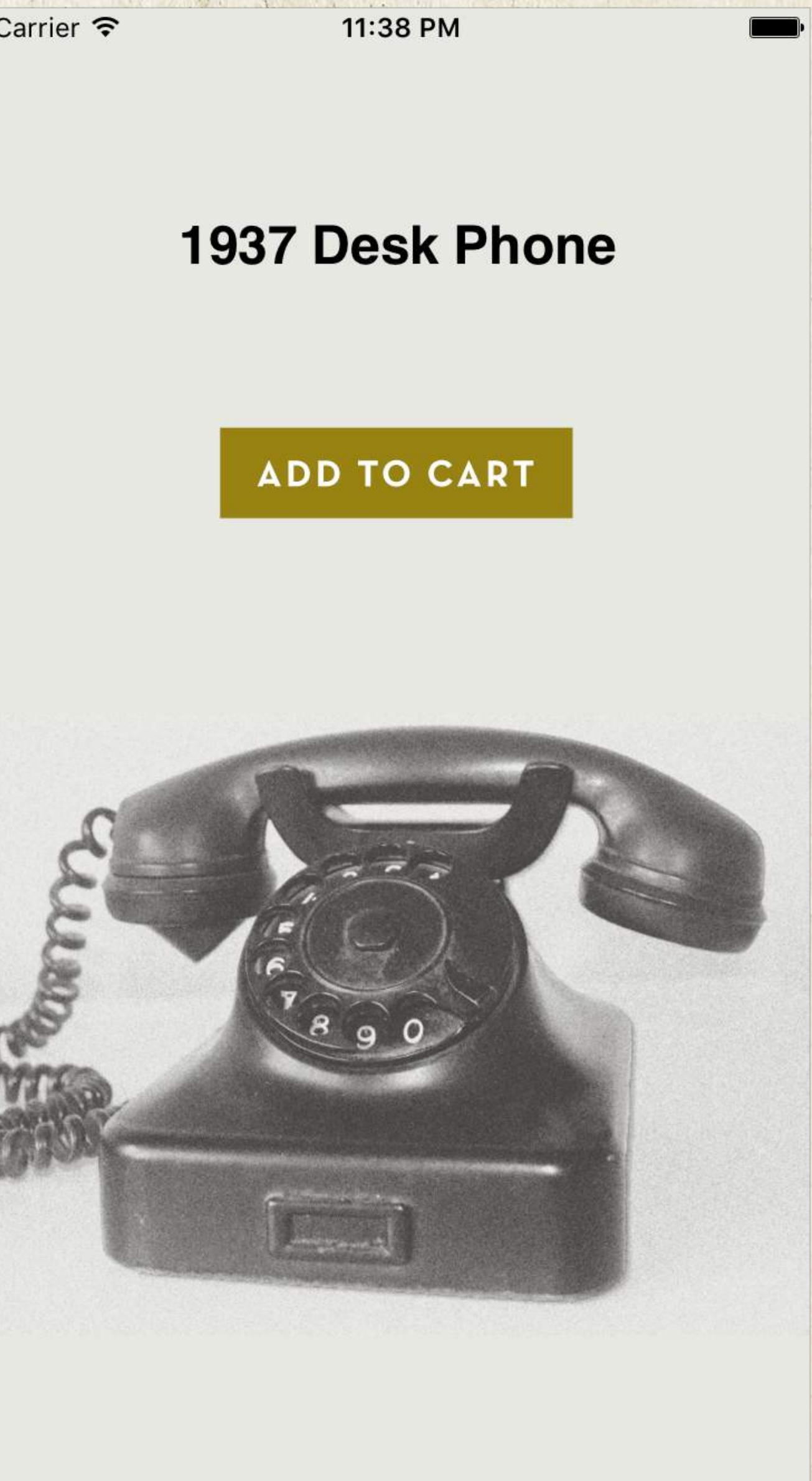
# Level 1

## Xcode and Storyboards

### Section 2 - Storyboards and Subviews



# Our First Goal: A Product Information Screen



APP  
EVOLUTION  
*with Swift*

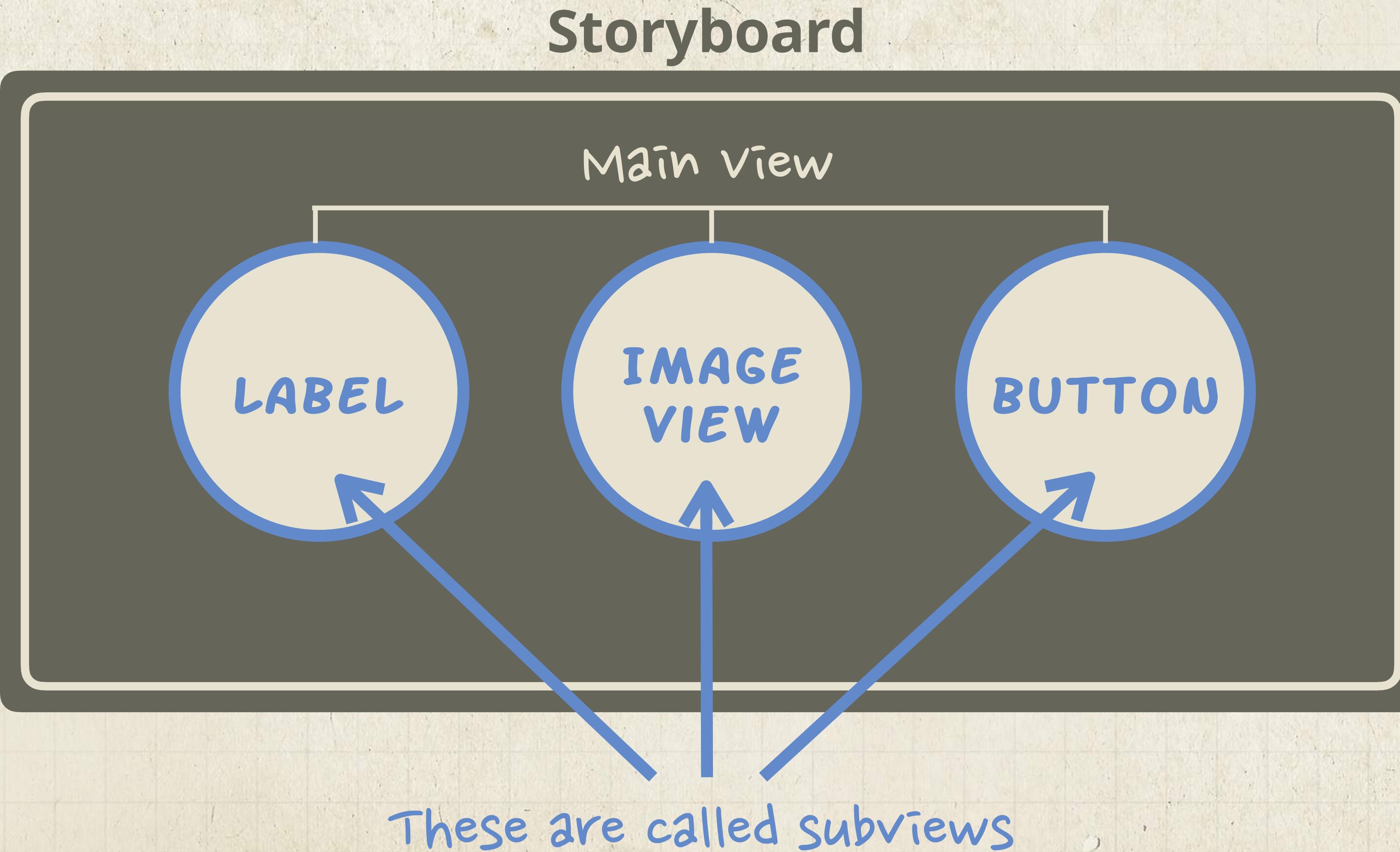
# Screencast: Linking a Swift File to the Storyboard

---



# AddingSubviews to the Main View

Let's add a label, image, and button to the main view.



# Screencast: Displaying and Styling a Label

---



# Screencast: Importing and Displaying an Image

---



# Screencast: Displaying and Styling a Button

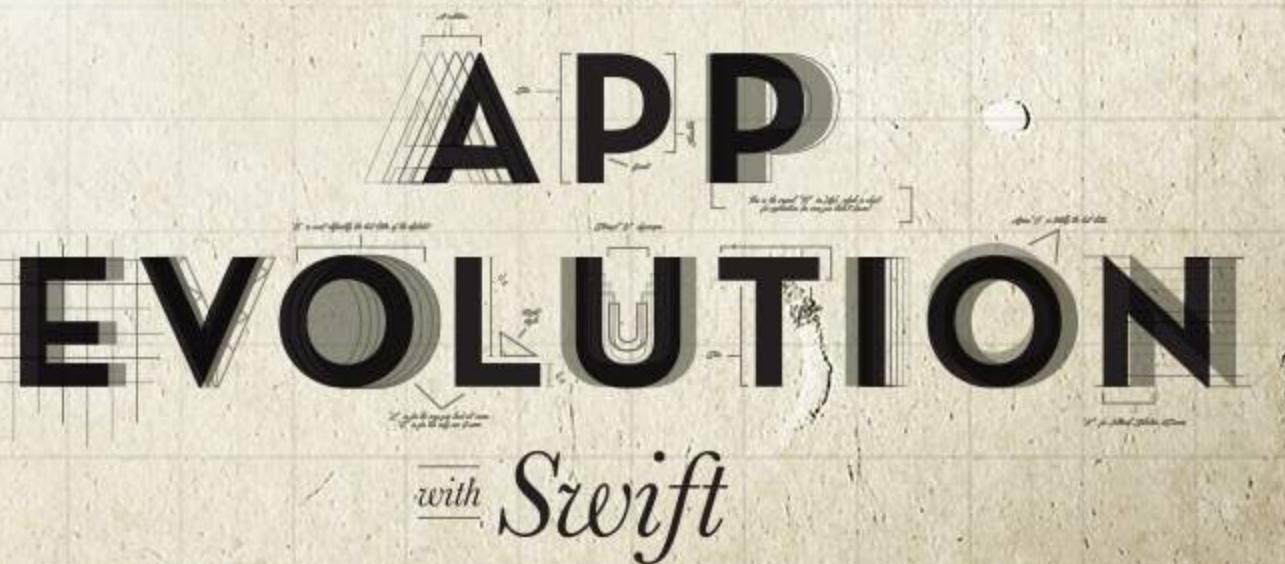
---



# Level 2

## Outlets and Actions

### Section 1 - Storyboard Outlets



# Problem: We Want to Set Content Programmatically

Here's what we have right now, but we want to change a few things.

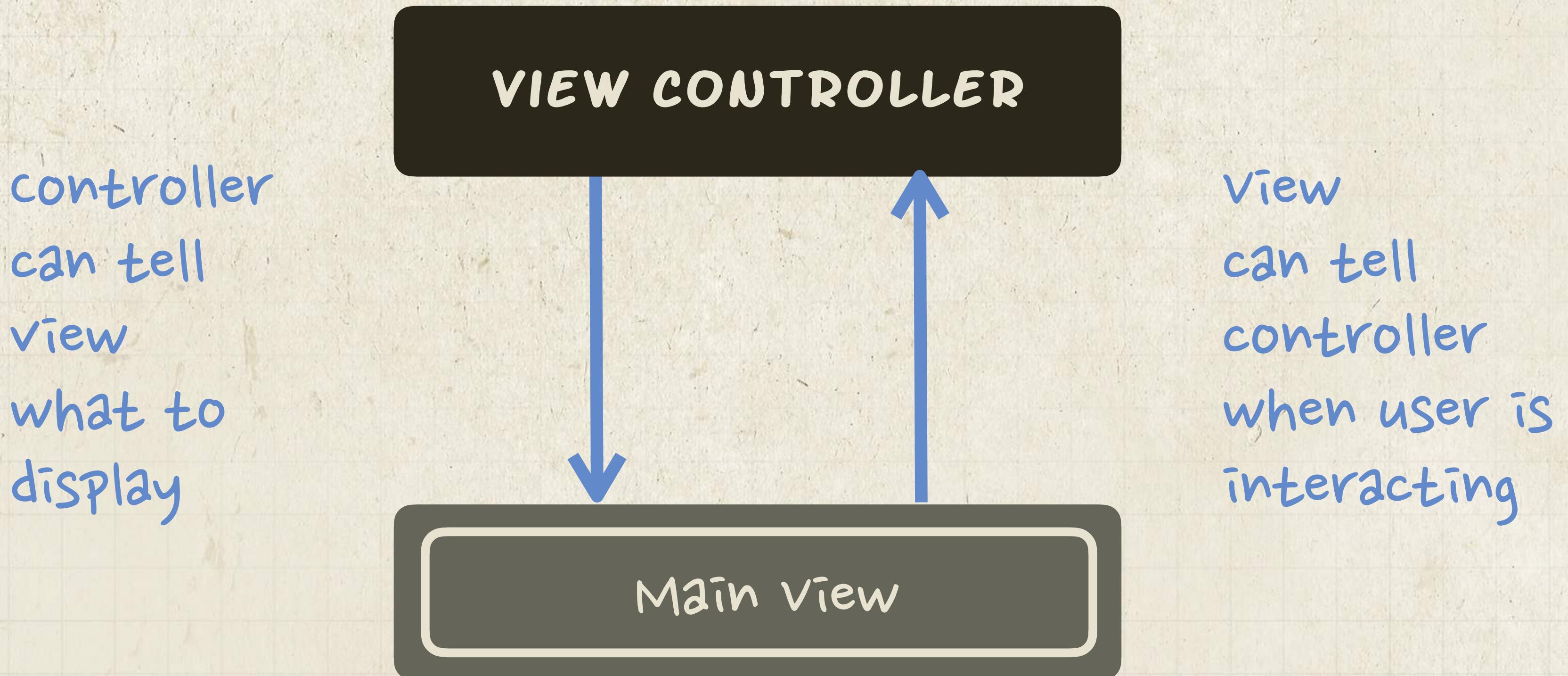
Set the text and image programmatically



Make this button do something when it is tapped

# Why View Controllers?

Separating the program logic from the display code makes the code you write easier to understand than 1 giant file.



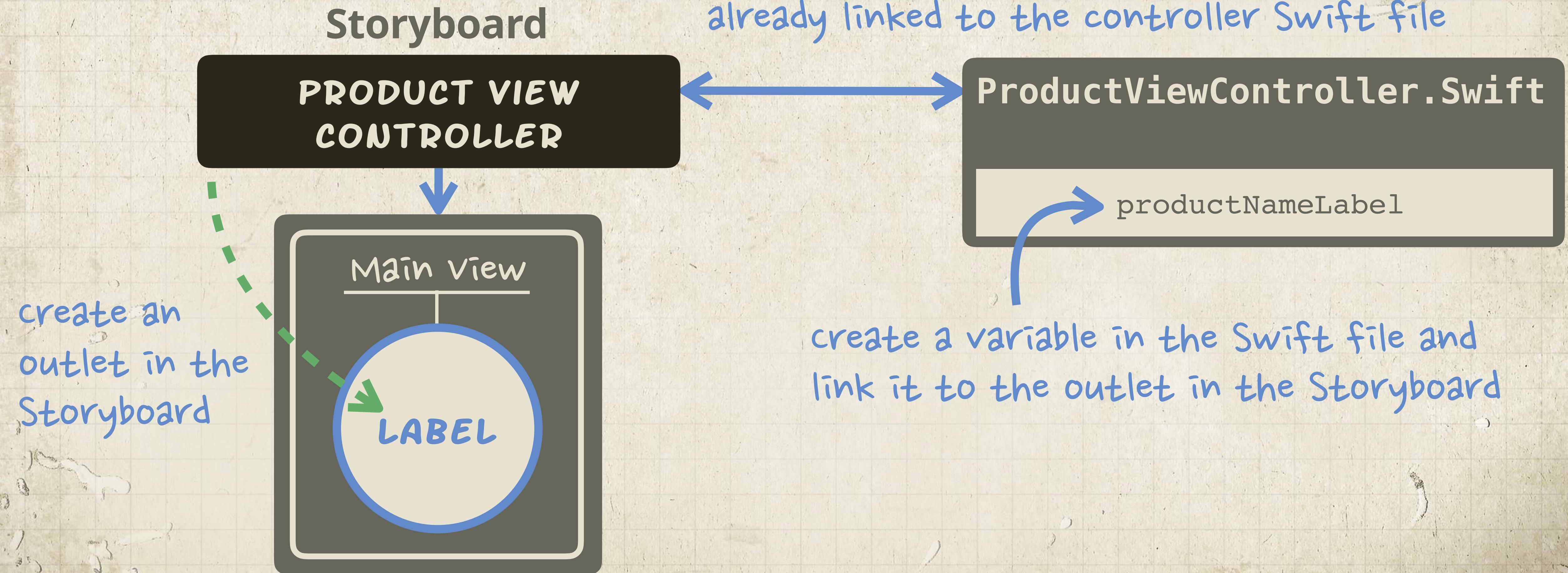
# Screencast: Creating a Swift File

---



# How Outlets Work

Outlets create a connection between a storyboard object and a variable in Swift.



# Screencast: Using the Assistant Editor

---



# The Controller After Connecting an Outlet

## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

    @IBOutlet weak var productNameLabel: UILabel!

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



Now we can access our Storyboard label in this file and change the text

But where should we write code to change that text?

# What Happens When an App Runs

---

App runs      The `AppDelegate.swift` file runs `UIApplicationMain()`  
which starts the app

App loads the storyboard      The Storyboard is set in the `Info.plist`  
configuration file

Storyboard loads the view controller      The view controller set as  
"initial" is the one that is loaded

View controller loads the view

Loading the view automatically runs the `viewDidLoad` function

# Setting the Label Text in Code

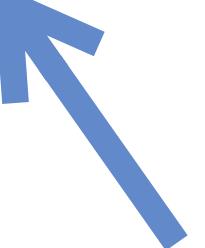
## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

    @IBOutlet weak var productNameLabel: UILabel!
    override func viewDidLoad() {
        super.viewDidLoad()
        productNameLabel.text = "1937 Desk Phone"
    }
}
```

This object is a UILabel



UILabels have a text property that you can set equal to a string

# Finding Properties in Apple's Documentation

## ProductViewController.swift

```
...  
    productNameLabel.text = "1937 Desk Phone"  
...
```

### Swift documentation for UILabel

#### text

The text displayed by the label.

#### Declaration

```
var text: String? { get set }
```

We can find the names of properties in Apple's documentation

Docs show properties and functions for each class

This means this property is expecting us to assign a string to it

Check out the full UIKit docs here:  
<http://go.codeschool.com/uikit-docs>

# Demo: Label Text Being Set with Code



APP  
EVOLUTION  
*with Swift*

# Screencast: Connecting the Image View to Code

---



# Setting the Image for an Image View With Code

## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

    @IBOutlet weak var productNameLabel: UILabel!
    @IBOutlet weak var productImageView: UIImageView!

    override func viewDidLoad() {
        super.viewDidLoad()

        productNameLabel.text = "1937 Desk Phone"
        productImageView.image = UIImage(named: "phone-fullscreen3")
    }
}
```

Image views have an `image` property that you can set equal to a `UIImage` object

# Creating a UIImage Object

```
UIImage(named: "phone-fullscreen3")
```

writing the name of a class with parentheses  
after it runs an initializer function

Swift documentation for UIImage

init(named:)

Returns the image object associated with the specified filename.

Declaration

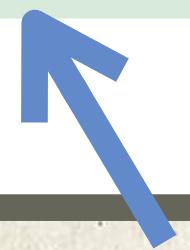
```
init?(named name: String)
```

This init function is expecting you to put a String  
with the named parameter in the parentheses

# Using the UIImage Object to Display an Image

ProductViewController.swift

```
...  
    productImageView.image = UIImage(named: "phone-fullscreen3")  
...
```

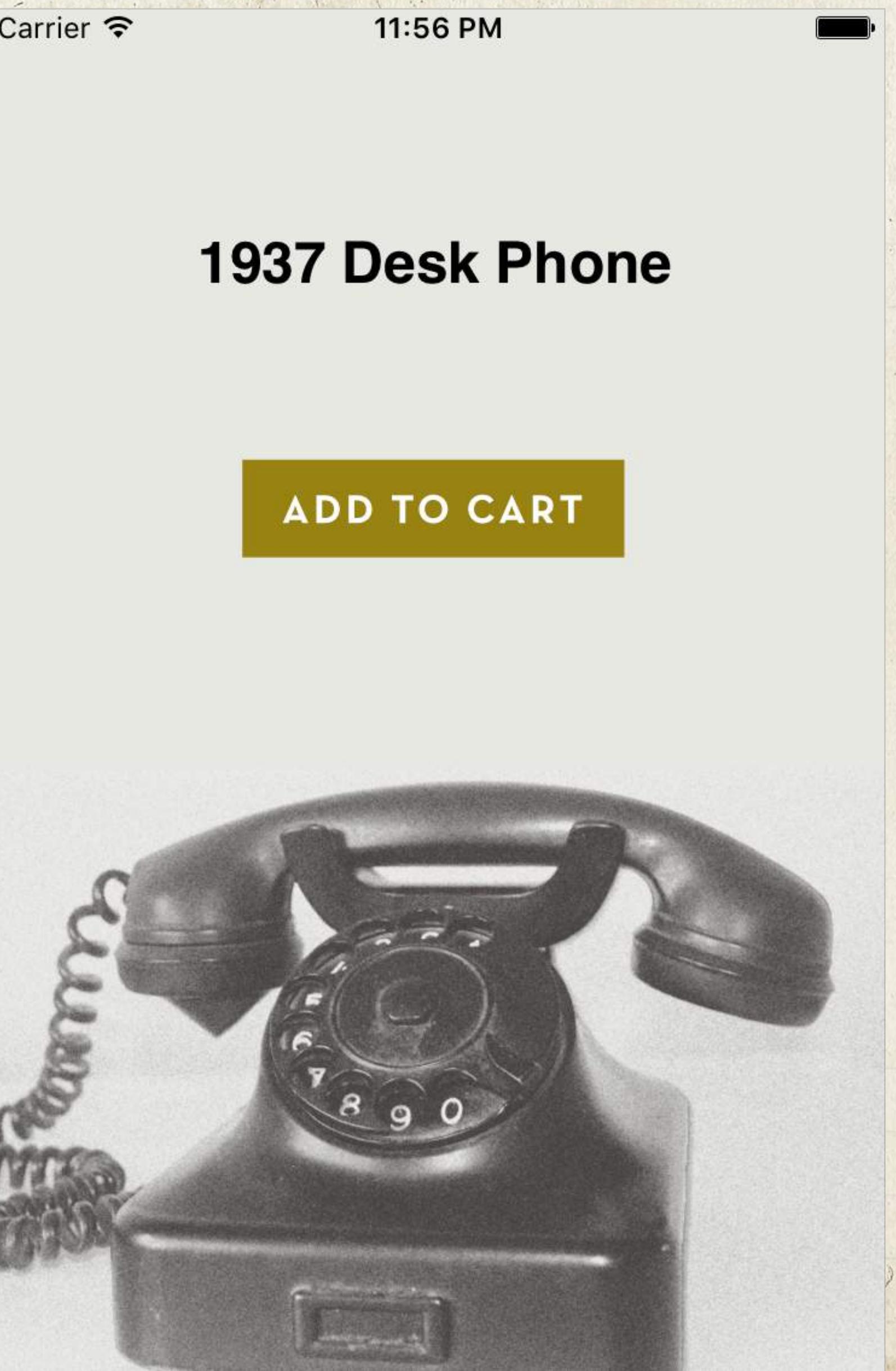


The **UIImage** initializer returns a **UIImage** object that the image view wants

# Screencast: Image Literals

---

# Demo: Image View Being Set With Code

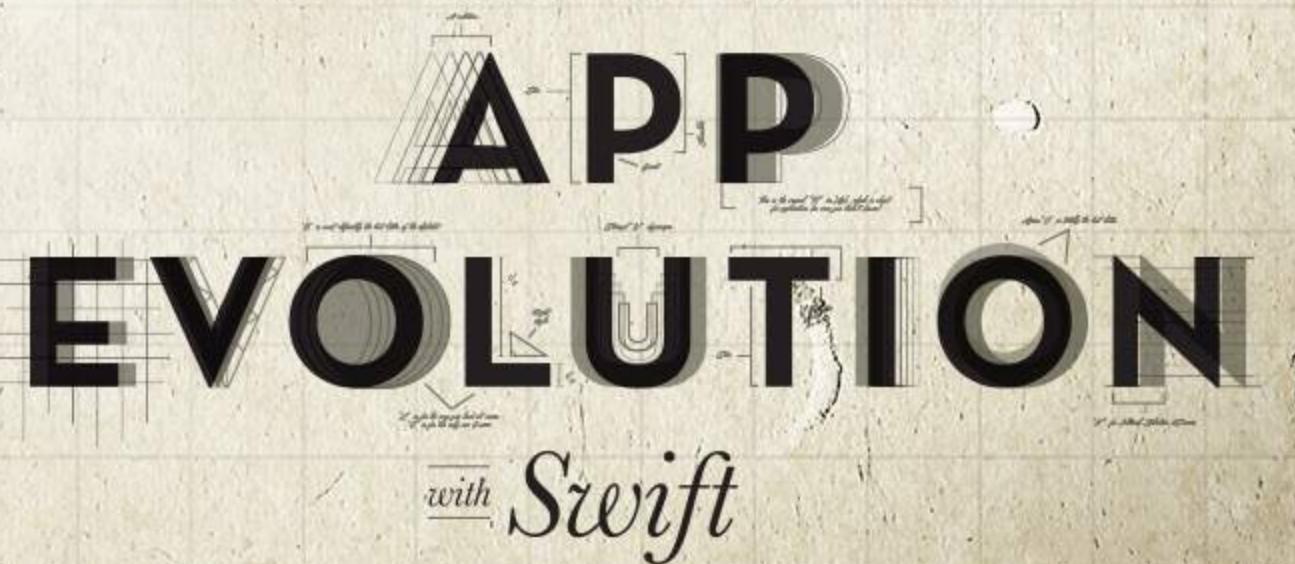


APP  
EVOLUTION  
*with Swift*

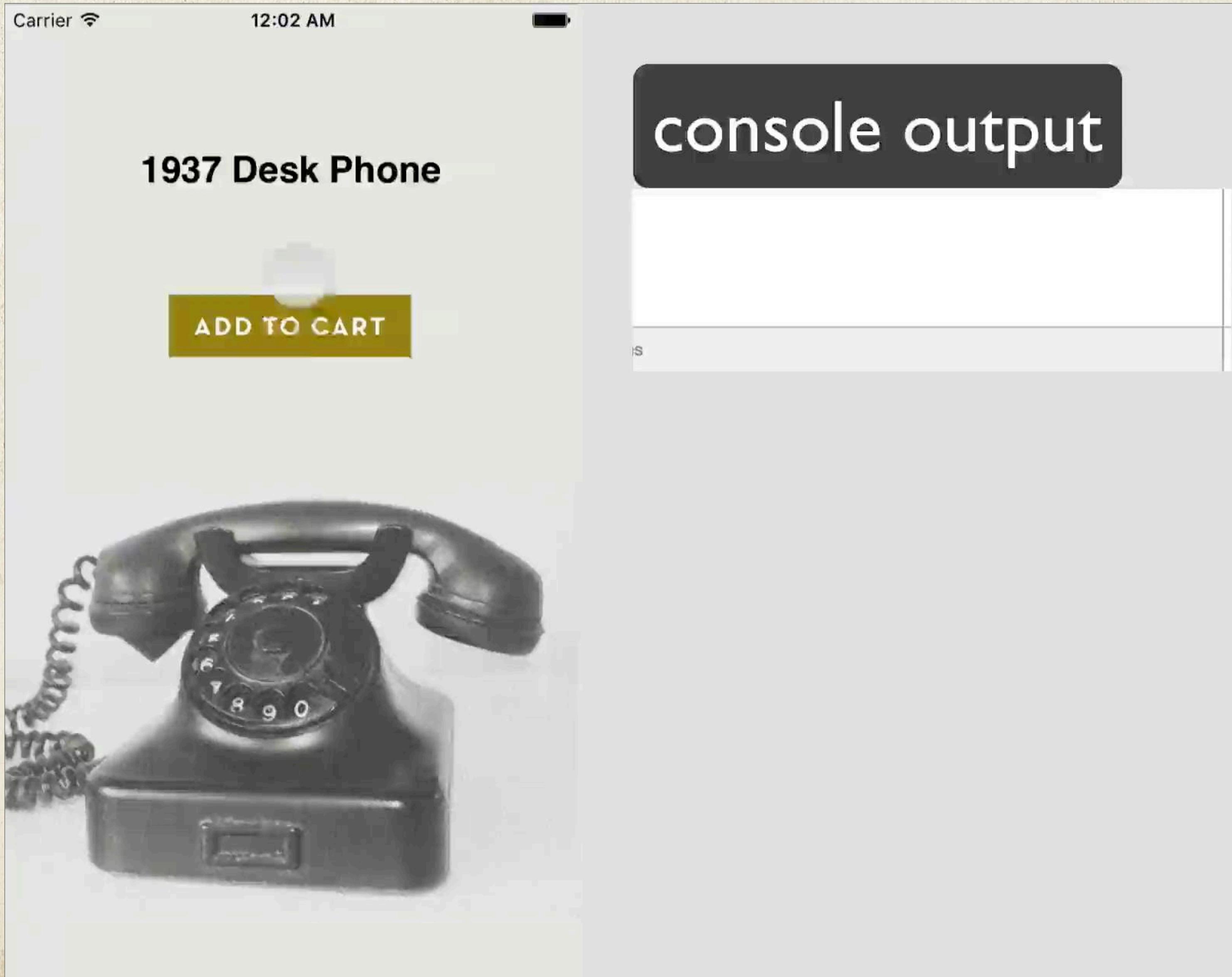
# Level 2

## Outlets and Actions

### Section 2 - Storyboard Actions

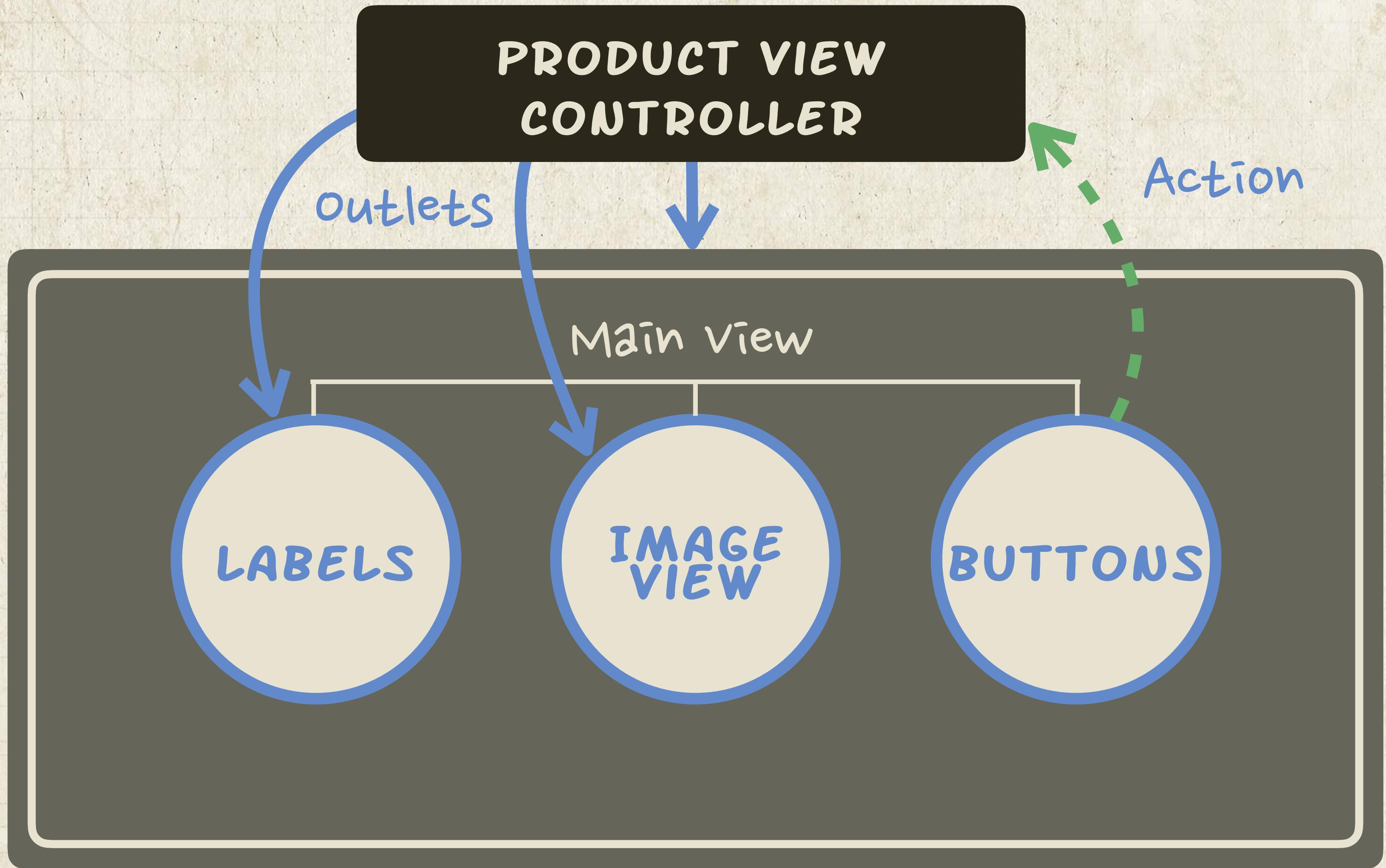


# Problem: Responding to an Action in a View



APP  
OLUTION  
*with Swift*

# Actions Are Created From Views to Controllers



Actions are used to send interaction information back to a view controller

# Screencast: Connect the Button With an Action

---



# Linking a Function to an Action

## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

    @IBOutlet weak var productNameLabel: UILabel!
    @IBOutlet weak var productImageView: UIImageView!

    override func viewDidLoad() { ... }

    @IBAction func addToCartPressed(_ sender: AnyObject) -> Void {
        }
}
```

This function now runs when the button is tapped

void means the function doesn't return any values



# Logging a Message When the Button Is Tapped

## ProductViewController.swift

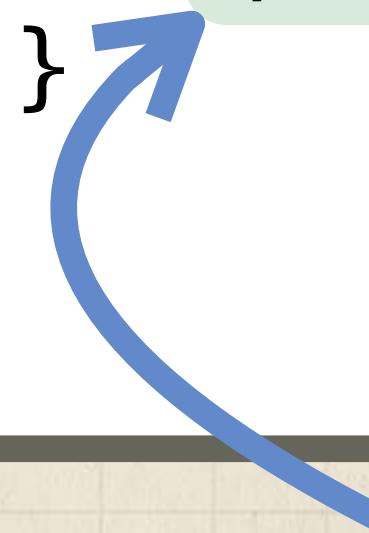
```
import UIKit

class ProductViewController: UIViewController {

    @IBOutlet weak var productNameLabel: UILabel!
    @IBOutlet weak var productImageView: UIImageView!

    override func viewDidLoad() { ... }

    @IBAction func addToCartPressed(_ sender: AnyObject) -> Void {
        print("Button tapped")
    }
}
```



print() logs a message to the Xcode console

# Demo: Button Tap Logs a Message to the Console

The screenshot shows the Xcode interface with the following details:

- Simulator Bar:** Shows "iPhone 6s - iPhone 6s / iOS 9.1 (13B137)" and "Carrier" status.
- File Navigator:** Displays the project structure for "GoodAsOldPhones".
- Main View:** Shows a product detail screen for a "1937 Desk Phone". It includes a black-and-white image of the phone, the text "1937 Desk Phone", and an "ADD TO CART" button.
- Code Editor:** Displays the "ProductViewController.swift" file content. The code defines a UIViewController subclass with properties for a product name label and image view. It also includes an "addPressed" method that logs a message to the console when the "ADD TO CART" button is tapped.

```
    controller: UIViewController {
        var productNameLabel: UILabel!
        var productImageView: UIImageView!
        override viewDidLoad() {
            super.viewDidLoad()
            self.load()
        }
        label.text = "1937 Desk Phone"
        view.image = UIImage(named: "phone-en3")
        addPressed(sender: AnyObject) {
            print("button tapped")
        }
    }
```

- Identity and Type Inspector:** Shows the file is named "ProductViewController.swift", has a type of "Default - Swift Source", and is located relative to the group.
- On Demand Resource Tags:** Shows "Only resources are laggable".
- Target Membership:** Shows the target is "GoodAsOldPhones".
- Text Settings:** Shows text encoding as "Unicode (UTF-8)", line endings as "Default - OS X / Unix (LF)", and indent using "Spaces".
- Documentation:** Provides descriptions for "Button", "Bar Button Item", and "Fixed Space Bar Button Item".

# Level 3

## Scroll Views



# Demo: Content Scrolling on the Screen

Carrier

12:04 PM

Fig. 2 Fig. 3 Fig. 4 Fig. 5

## ABOUT US

Good as Old Phones returns the phones of yesteryear back to their original glory and then gets them into the hands\* of those who appreciate them most.

Whether you're looking for a turn-of-the-century wall set or a Zack Morris special, we've got you covered. Give us a ring, and we'll get you connected.

\*Hands-free phones available

## CONTACT

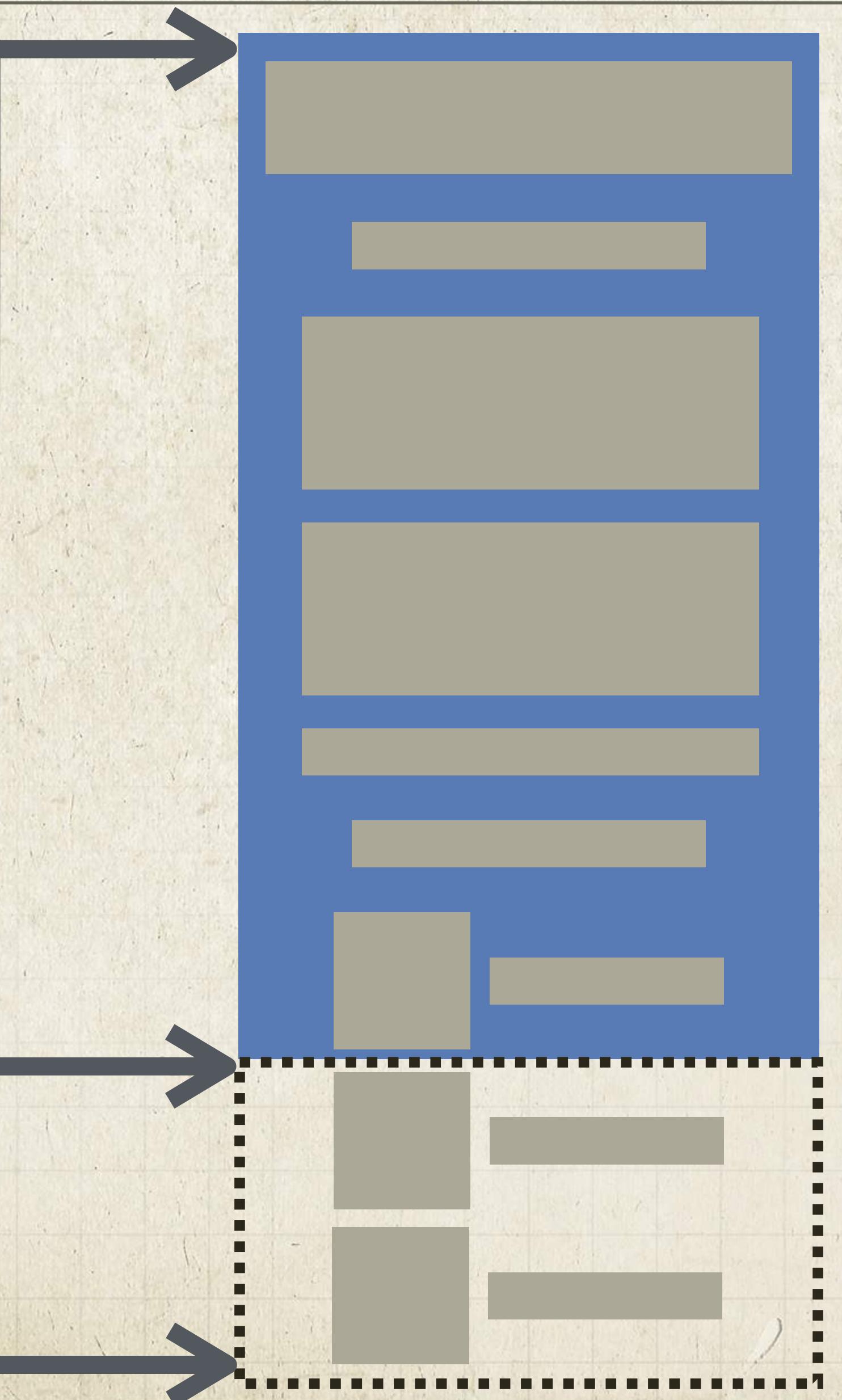
good-as-old@example.com

APP  
EVOLUTION  
with Swift

# How a Scroll View Works

The available space  
on the iPhone screen

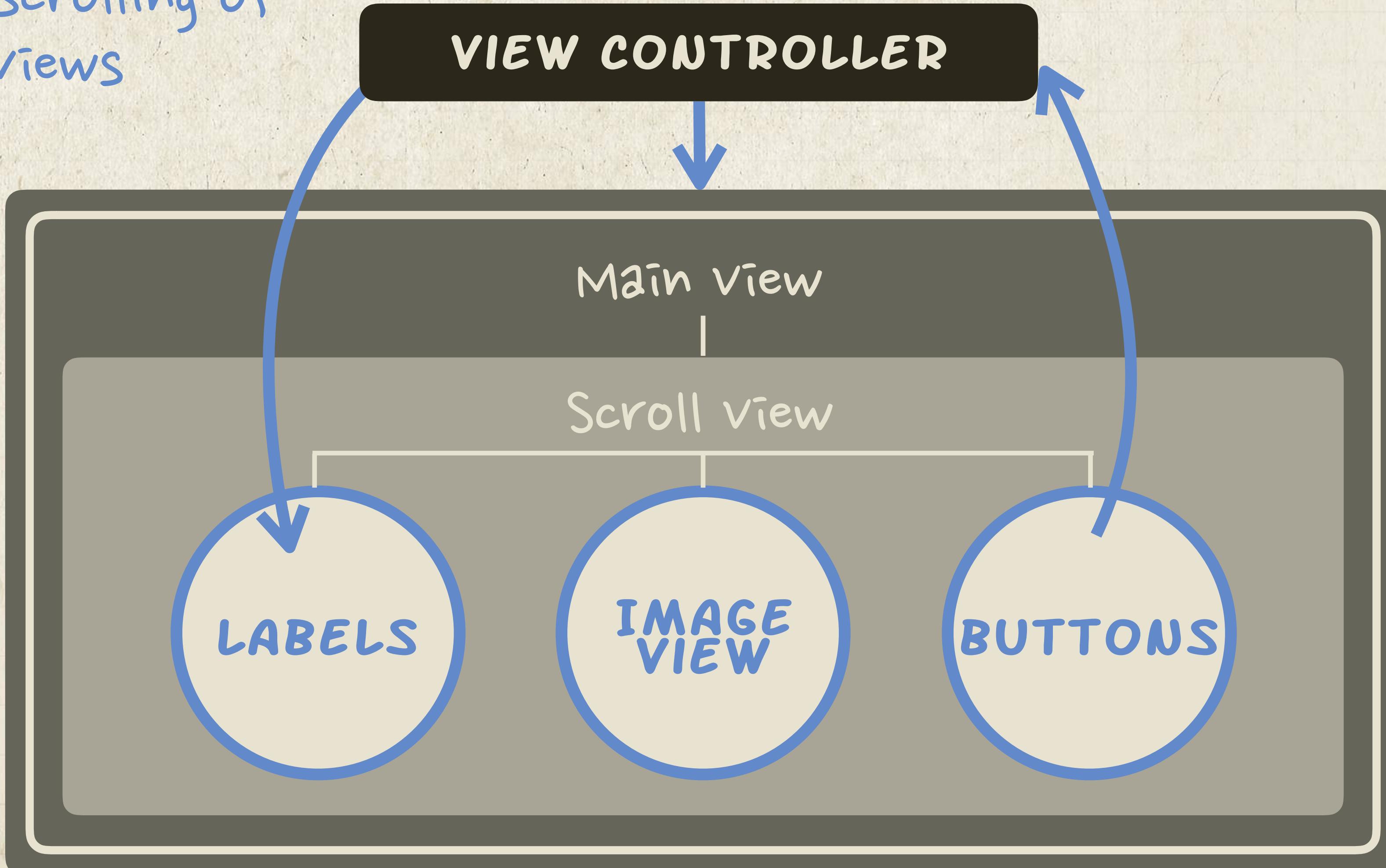
Sometimes your  
content needs more  
space than that



APP  
EVOLUTION  
with Swift

# Scroll Views Contain Any Subviews That Can Be Scrolled

A `UIScrollView` object  
manages the scrolling of  
all of its subviews



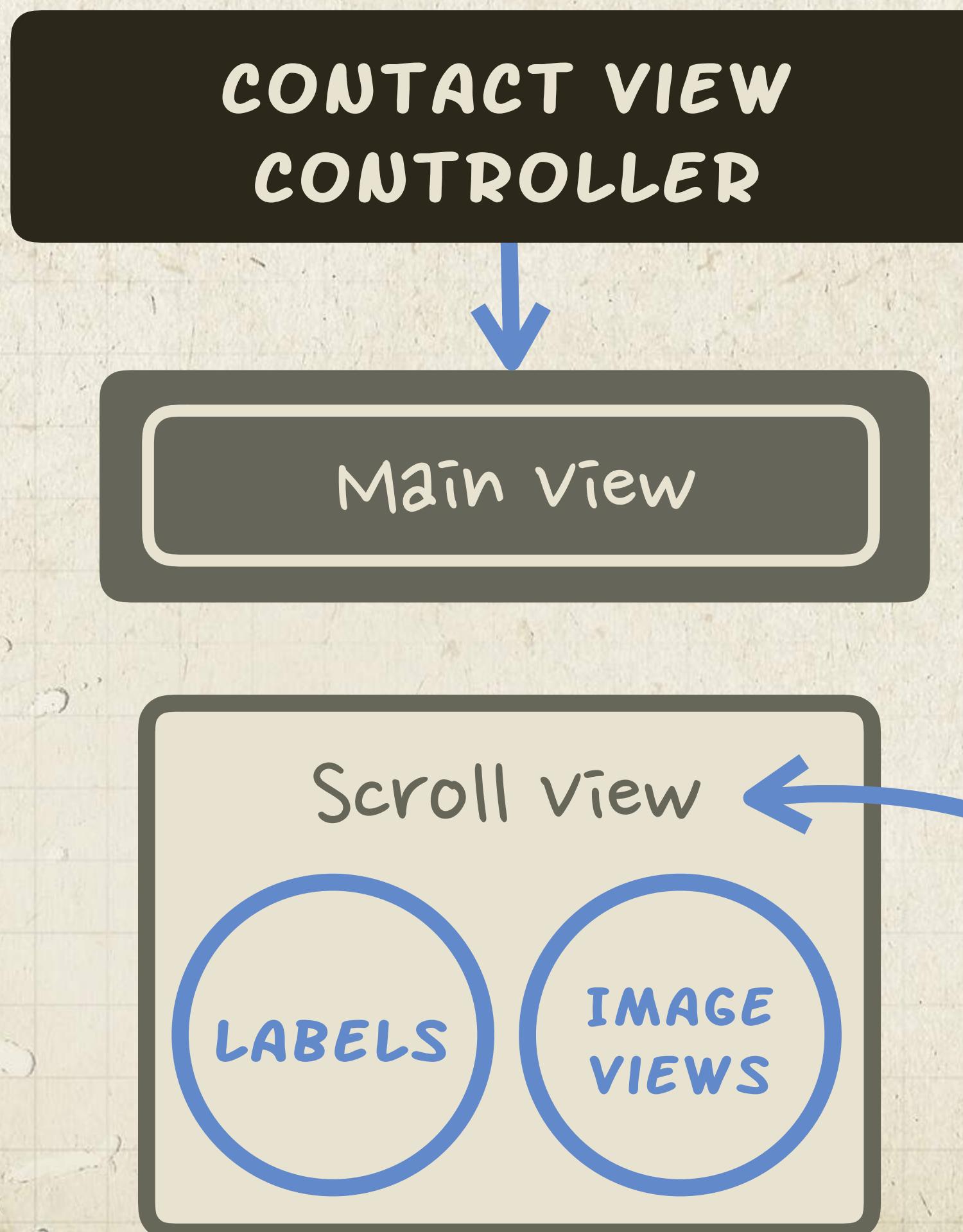
# Screencast: Creating a Scroll View in the Storyboard

---



# Problem: Scroll View Isn't Connected

The scroll view in our storyboard scene isn't a subview of the main view, so we need to manually set that in code.



The scroll view isn't  
connected to anything  
right now!

# Connecting the Scroll View to the Main View

## ContactViewController.swift

```
class ContactViewController: UIViewController {  
  
    @IBOutlet weak var scrollView: UIScrollView!  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        view.addSubview(scrollView)  
    }  
}
```

This makes the scroll view part of the main view

CONTACT  
VIEW  
CONTROLLER



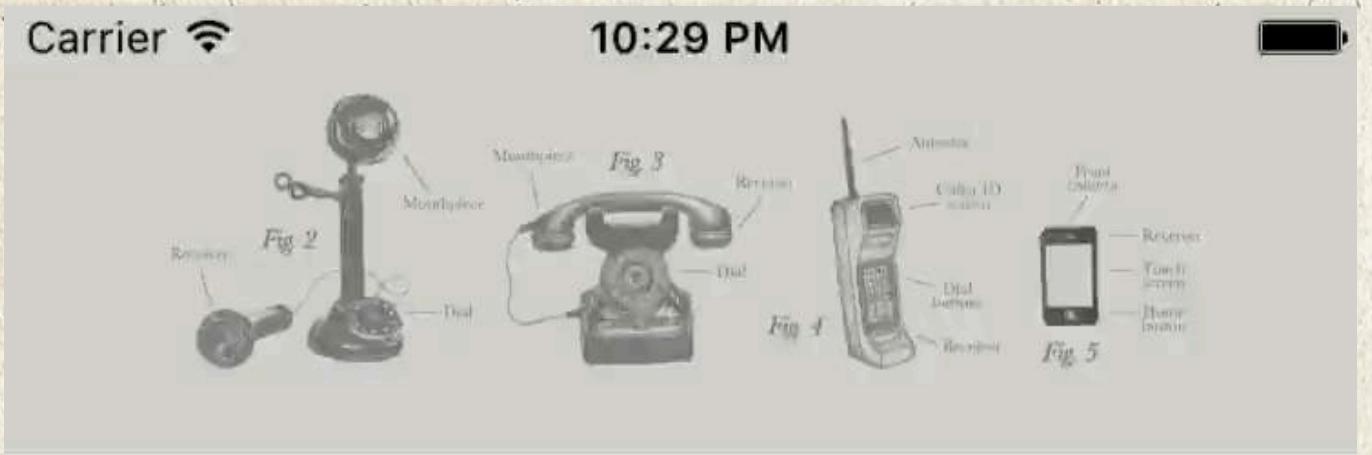
Main view

Scroll view

LABELS

IMAGE  
VIEWS

# Problem: The Scroll View Isn't Scrolling



## ABOUT US

Good as Old Phones returns the phones of yesteryear back to their original glory and then gets them into the hands\* of those who appreciate them most.

Whether you're looking for a turn-of-the-century wall set or a Zack Morris special, we've got you covered. Give us a ring, and we'll get you connected.

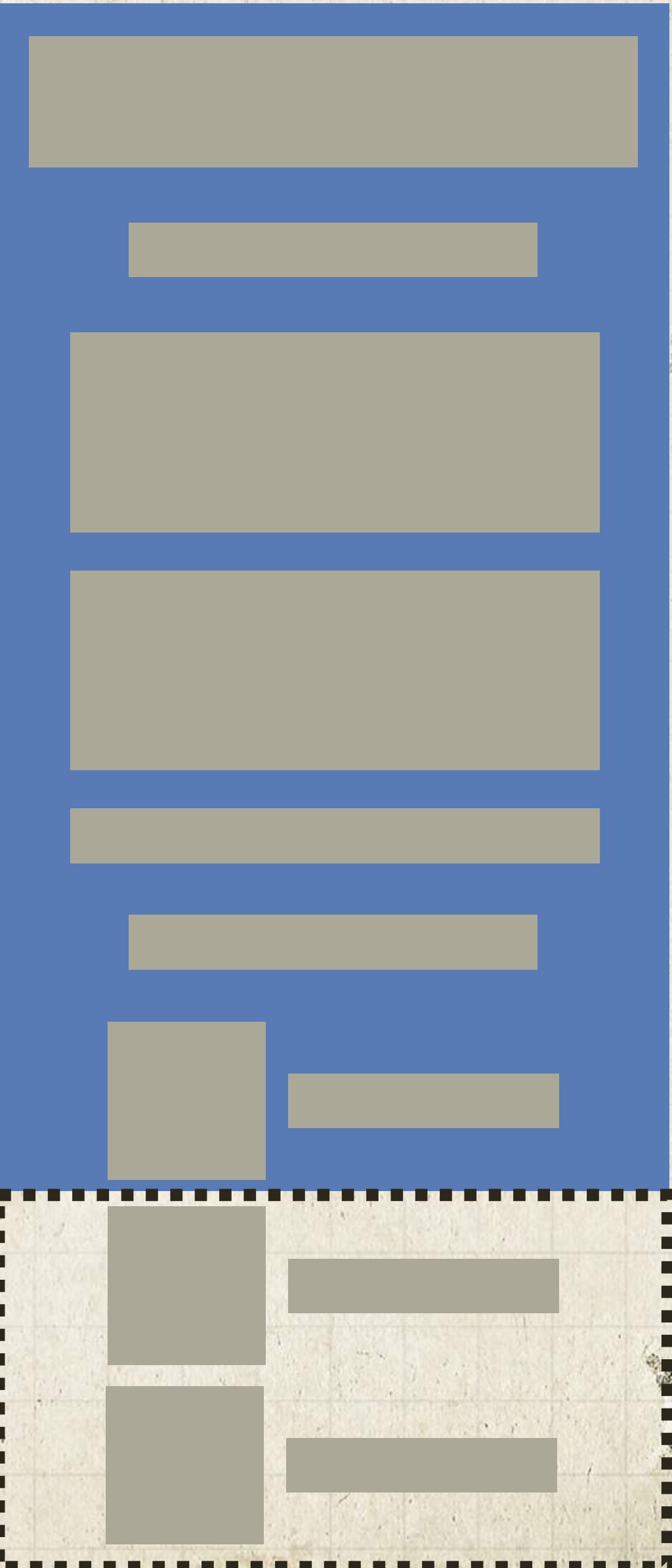
*\*Hands-free phones available*

## CONTACT



good-as-old@example.com

Need to tell the scroll view the height and width of the extra content it contains

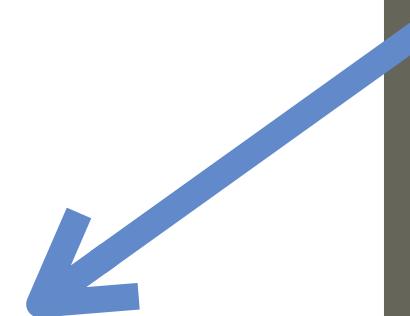


# Adding a Function That Runs After viewDidLoad

## ContactViewController.swift

```
class ContactViewController: UIViewController {  
  
    @IBOutlet weak var scrollView: UIScrollView!  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        view.addSubview(scrollView)  
    }  
  
    override func viewWillLayoutSubviews() {  
        super.viewWillLayoutSubviews()  
    }  
}
```

Any time you're changing the size of a subview, do it in this function



# Setting the Scroll View's Content Size

## ContactViewController.swift

```
class ContactViewController: UIViewController {  
  
    @IBOutlet weak var scrollView: UIScrollView!  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        view.addSubview(scrollView)  
    }  
  
    override func viewWillLayoutSubviews() {  
        super.viewWillLayoutSubviews()  
  
        scrollView.contentSize = CGSize(width: 375, width: 800)  
    }  
}
```

This number  
should be big  
enough to hold all  
of the content



# Demo: The Working Scroll View

Carrier ⌘ 12:04 PM

Fig. 2 Fig. 3 Fig. 4 Fig. 5

**ABOUT US**

Good as Old Phones returns the phones of yesteryear back to their original glory and then gets them into the hands\* of those who appreciate them most.

Whether you're looking for a turn-of-the-century wall set or a Zack Morris special, we've got you covered. Give us a ring, and we'll get you connected.

\*Hands-free phones available

**CONTACT**

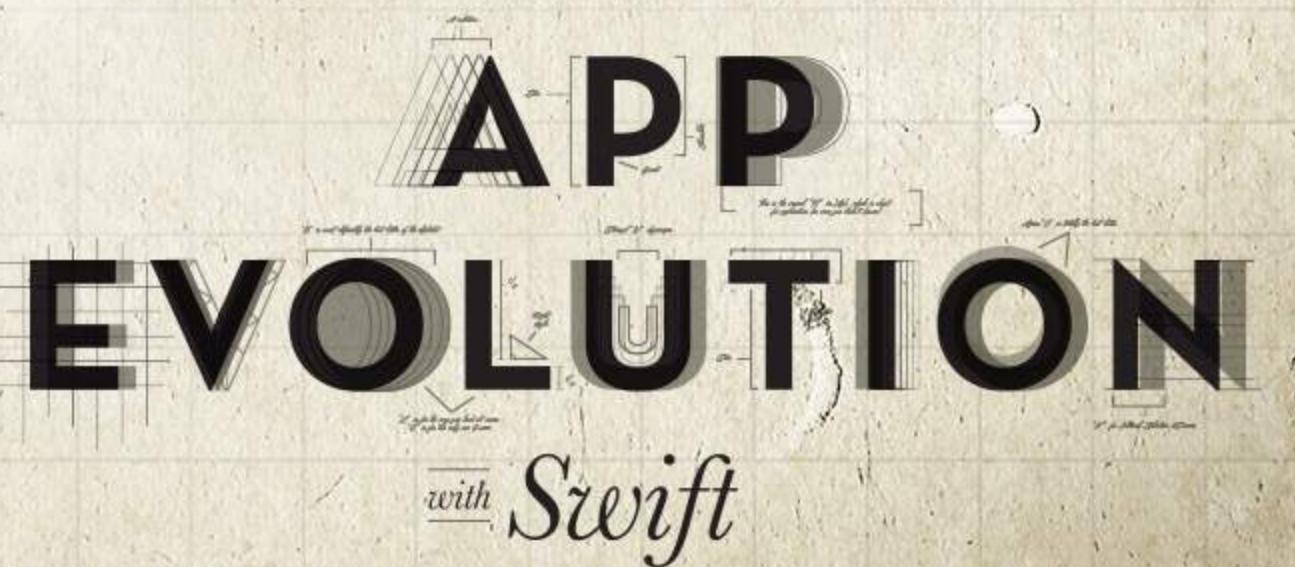
good-as-old@example.com

APP  
EVOLUTION  
with Swift

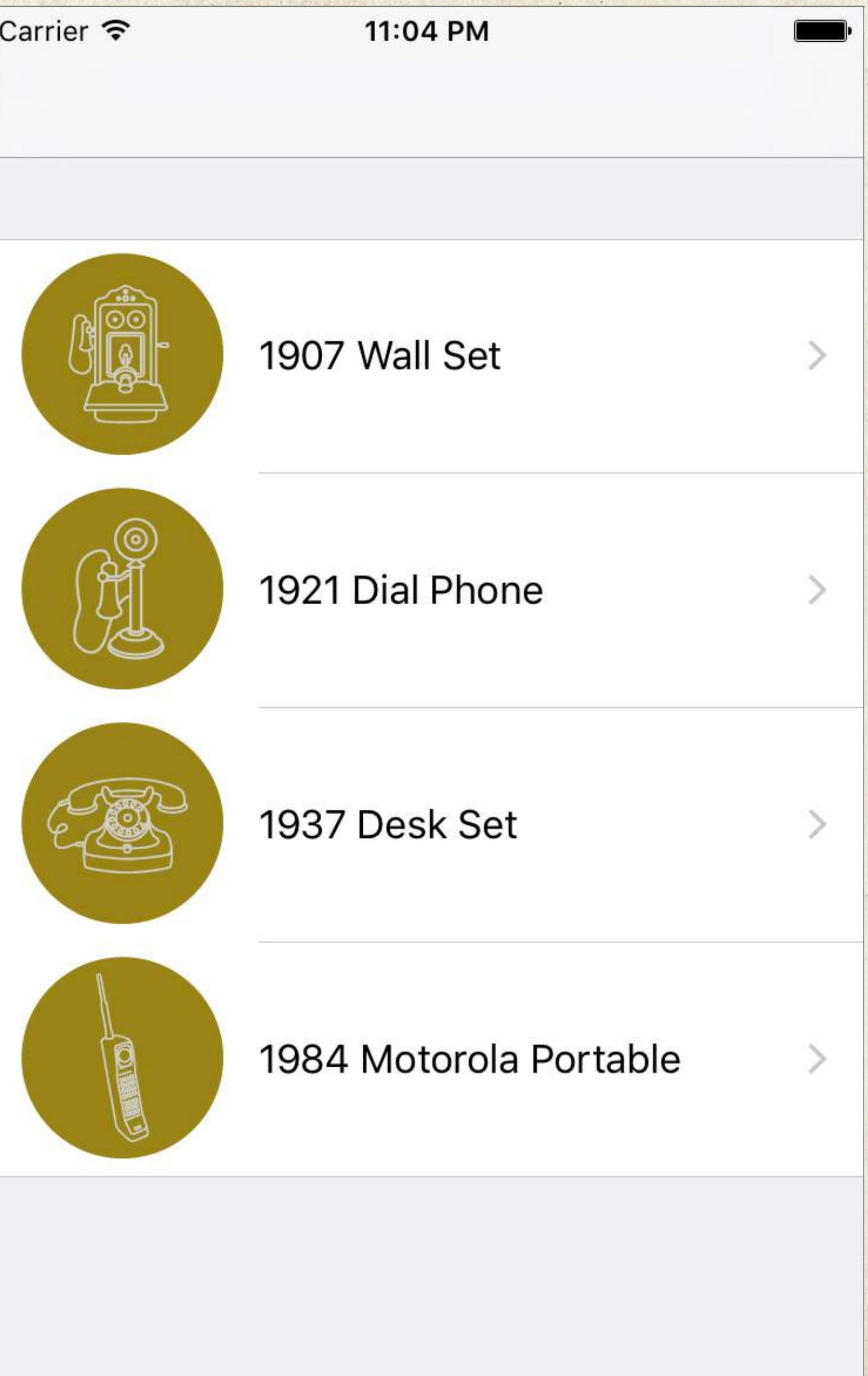
# Level 4

## Table Views

### Section 1 - Creating a Table View



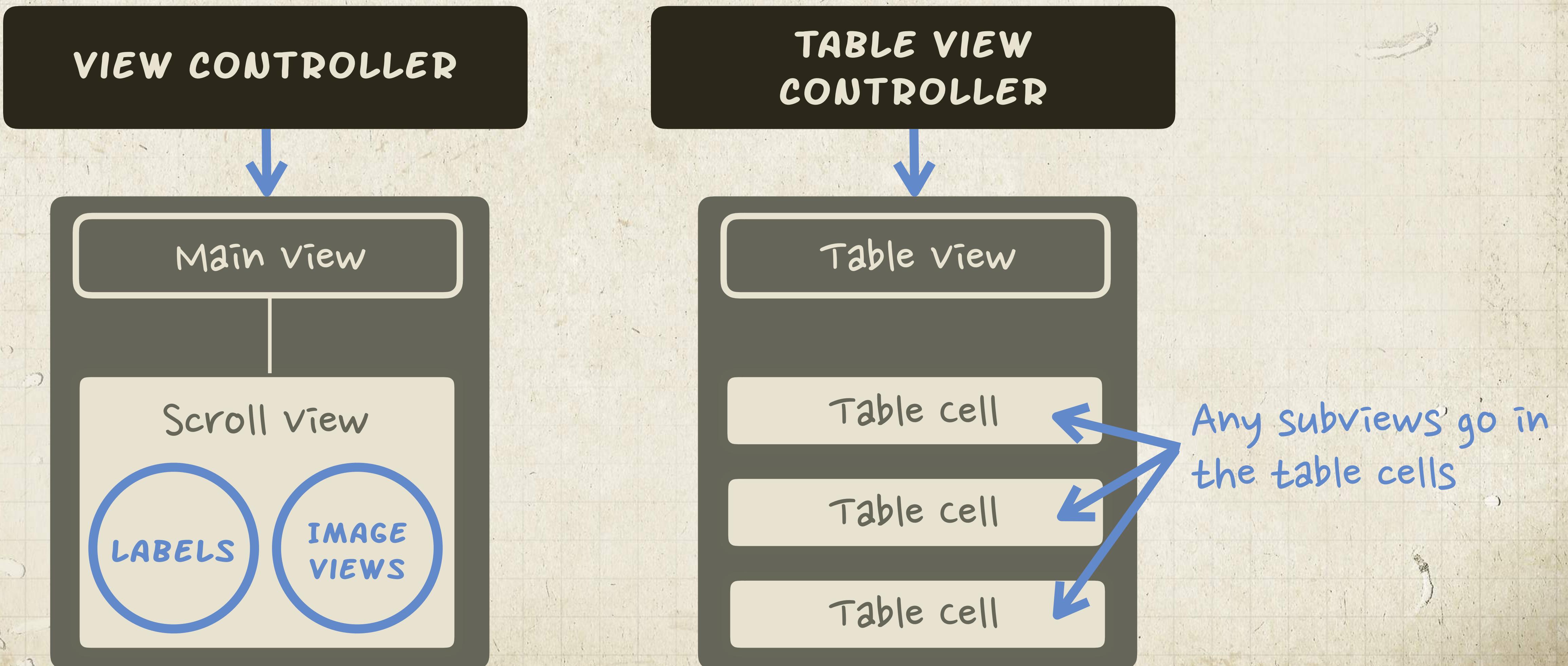
# Demo: Using a Table View to Display a List of Things



APP  
EVOLUTION  
*with Swift*

# A Table View Controller's Hierarchy

A Table View Controller is just a special View Controller that has support for working with Table Views built-in



# Screencast: Creating a Table View Controller

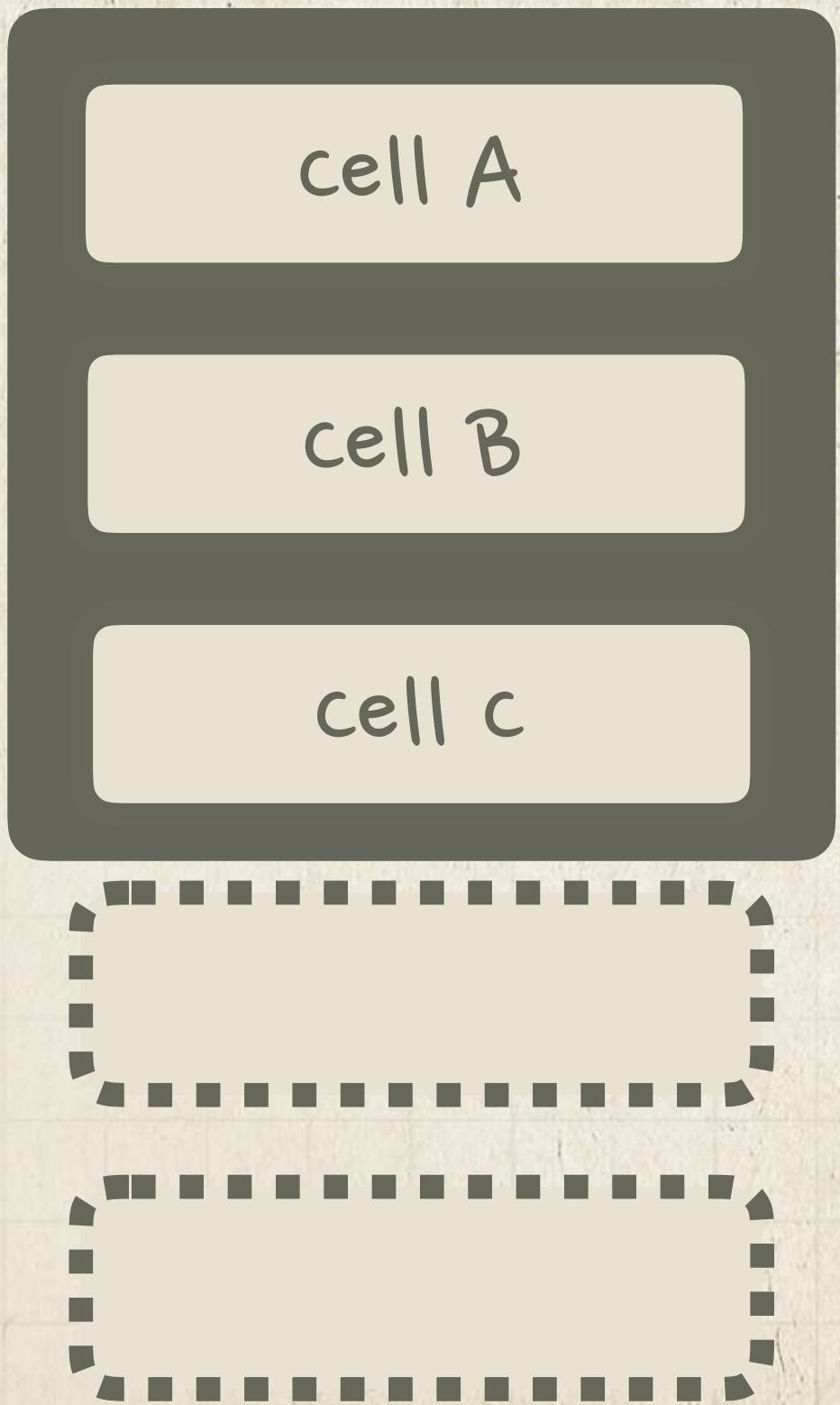
---



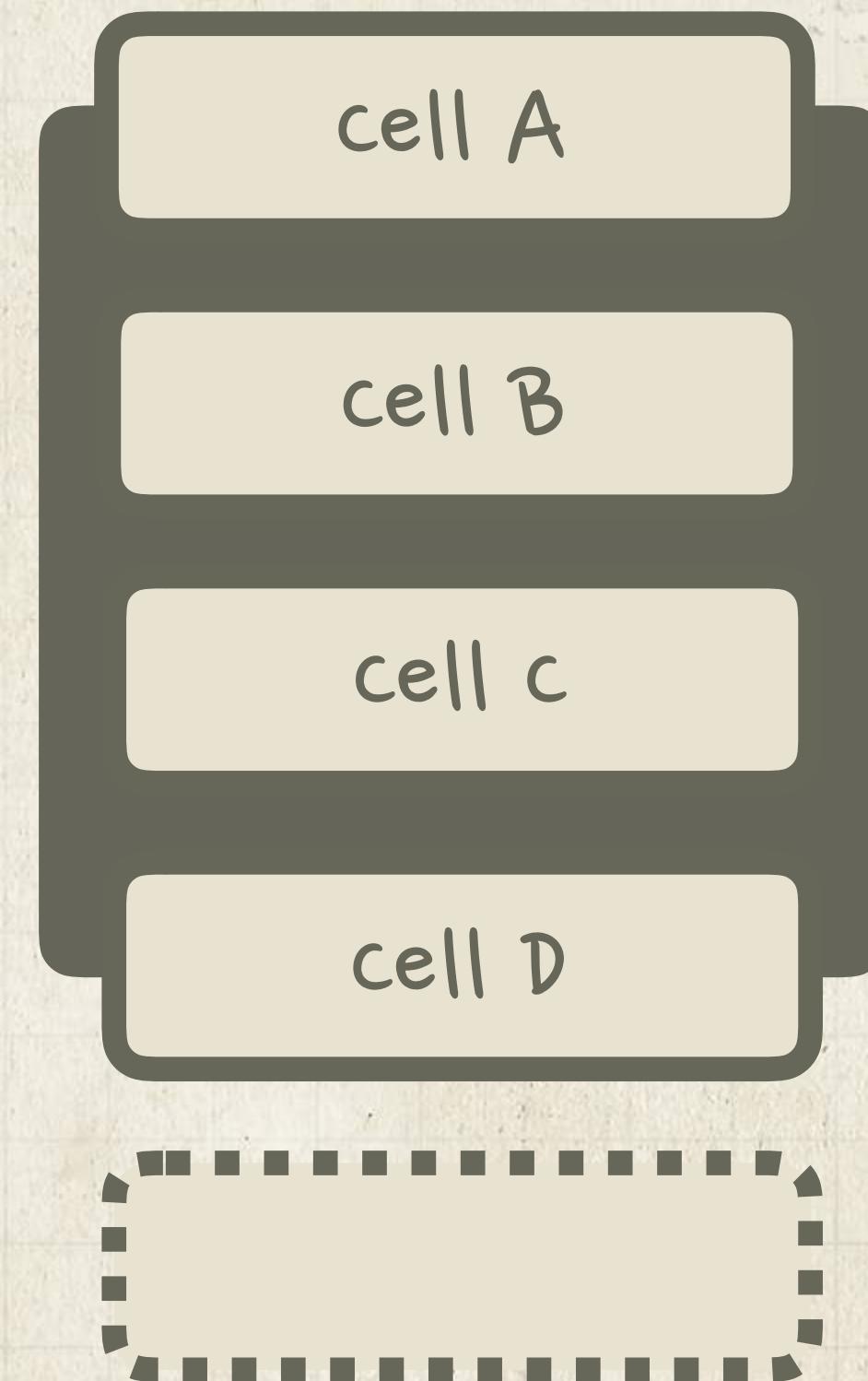
# How Table View Cell Recycling Works

Table cells are recycled as they scroll offscreen, and any new cells are just recycled cells filled in with new data.

Any visible cells are created on the initial load of the table view

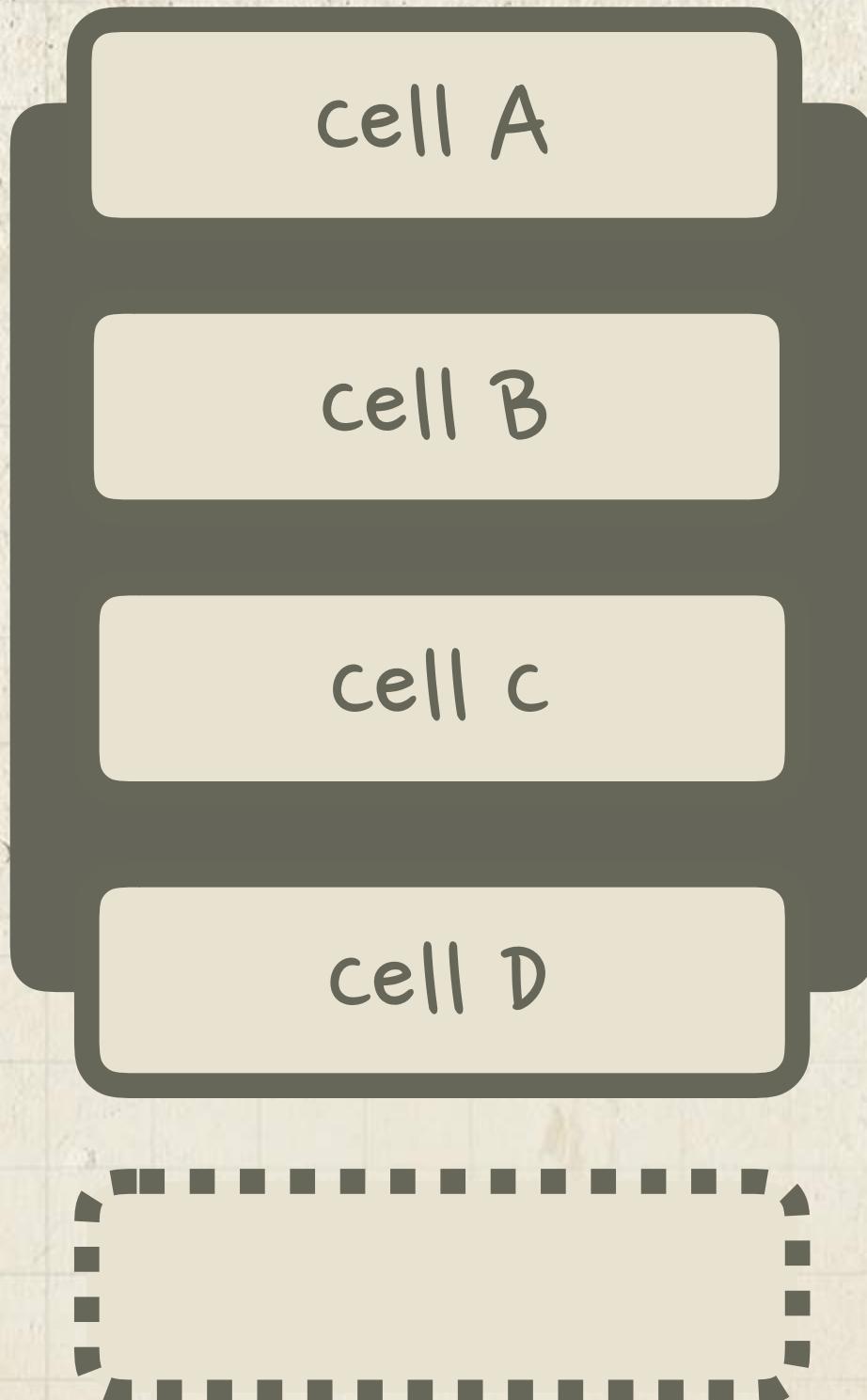


As the table view scrolls, now four unique cells are visible

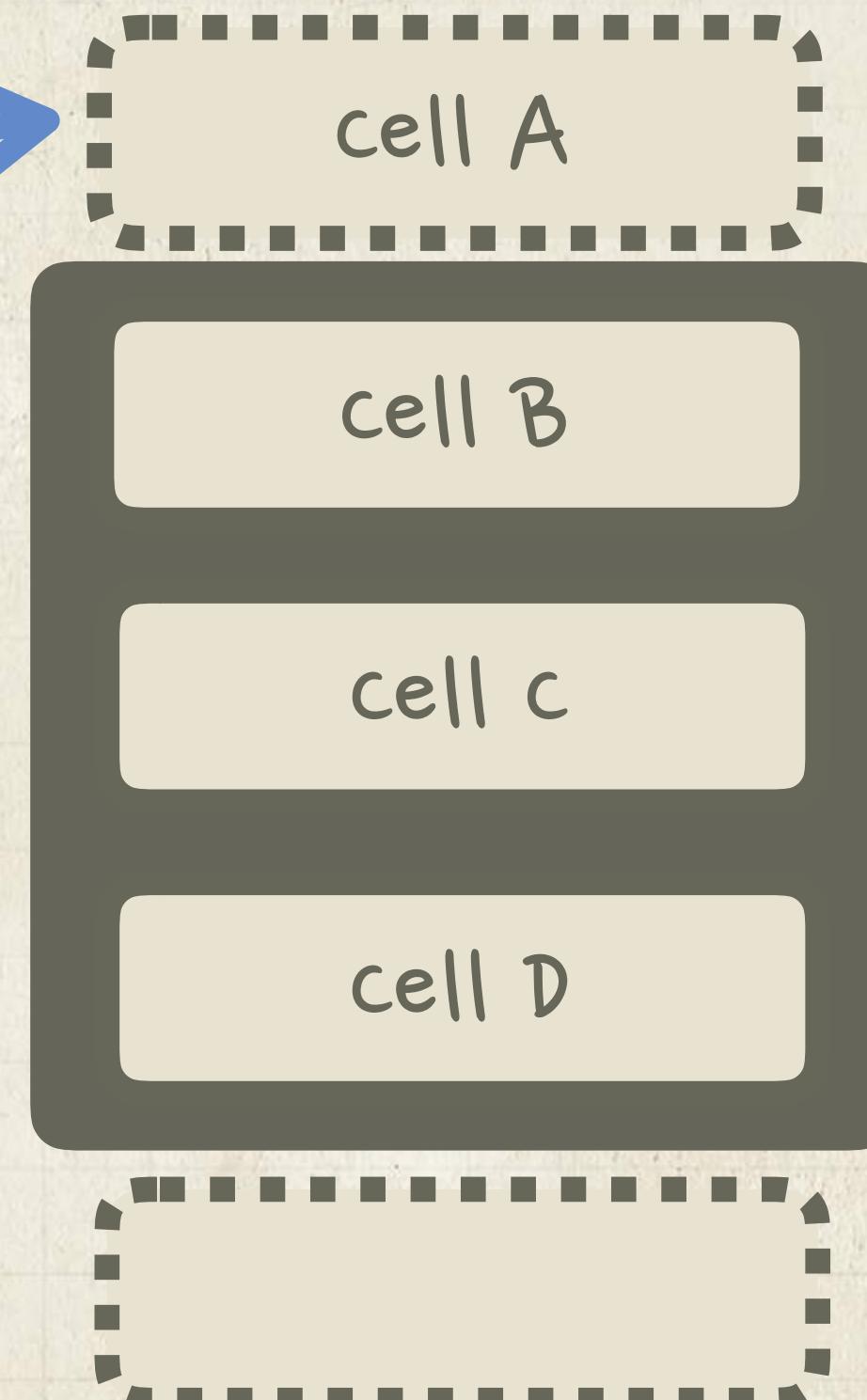


# How Table View Cell Recycling Works

Table cells are recycled as they scroll offscreen, and any new cells are just recycled cells filled in with new data.

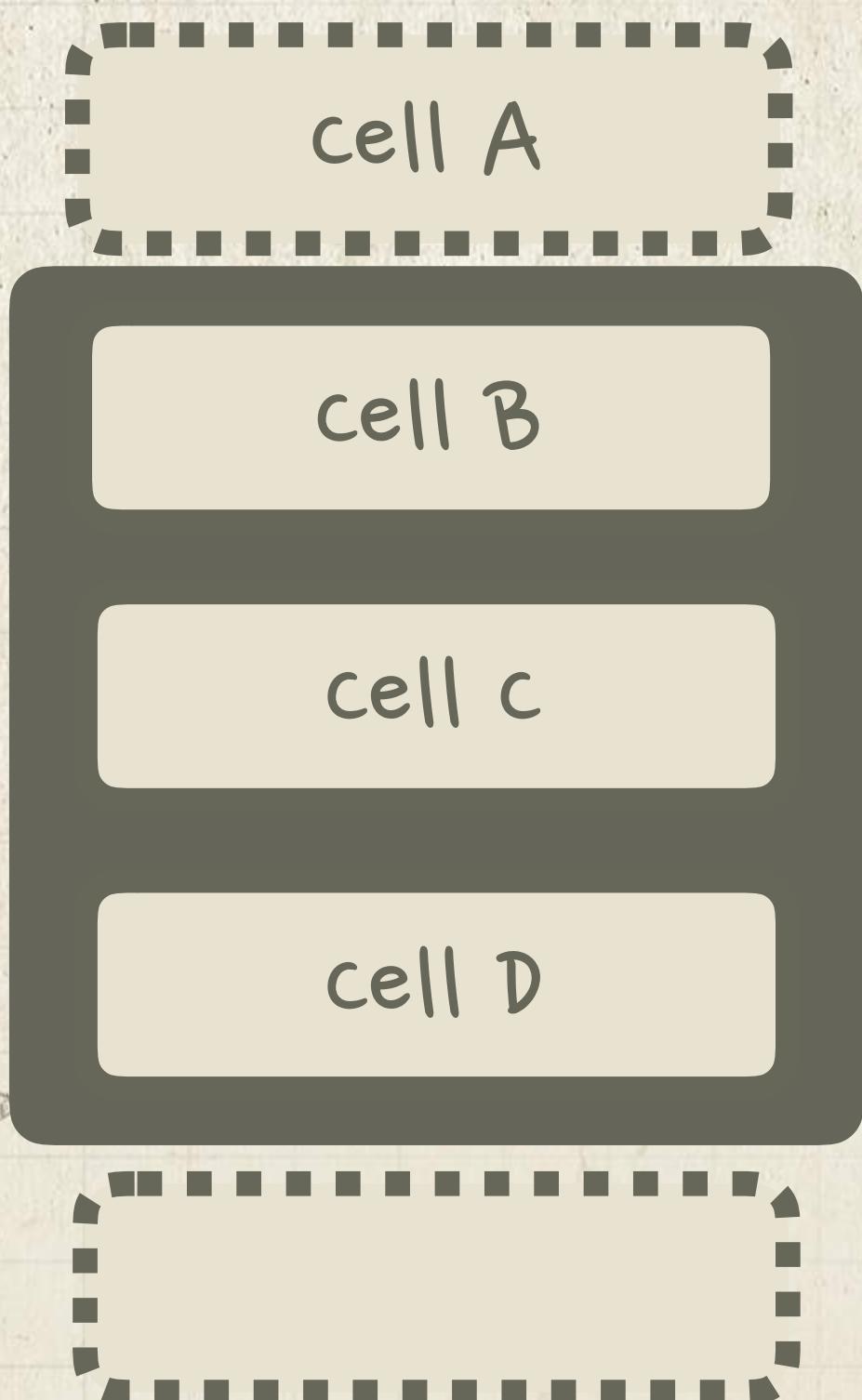


After scrolling some  
more, cell A is no  
longer visible and  
marked as recyclable

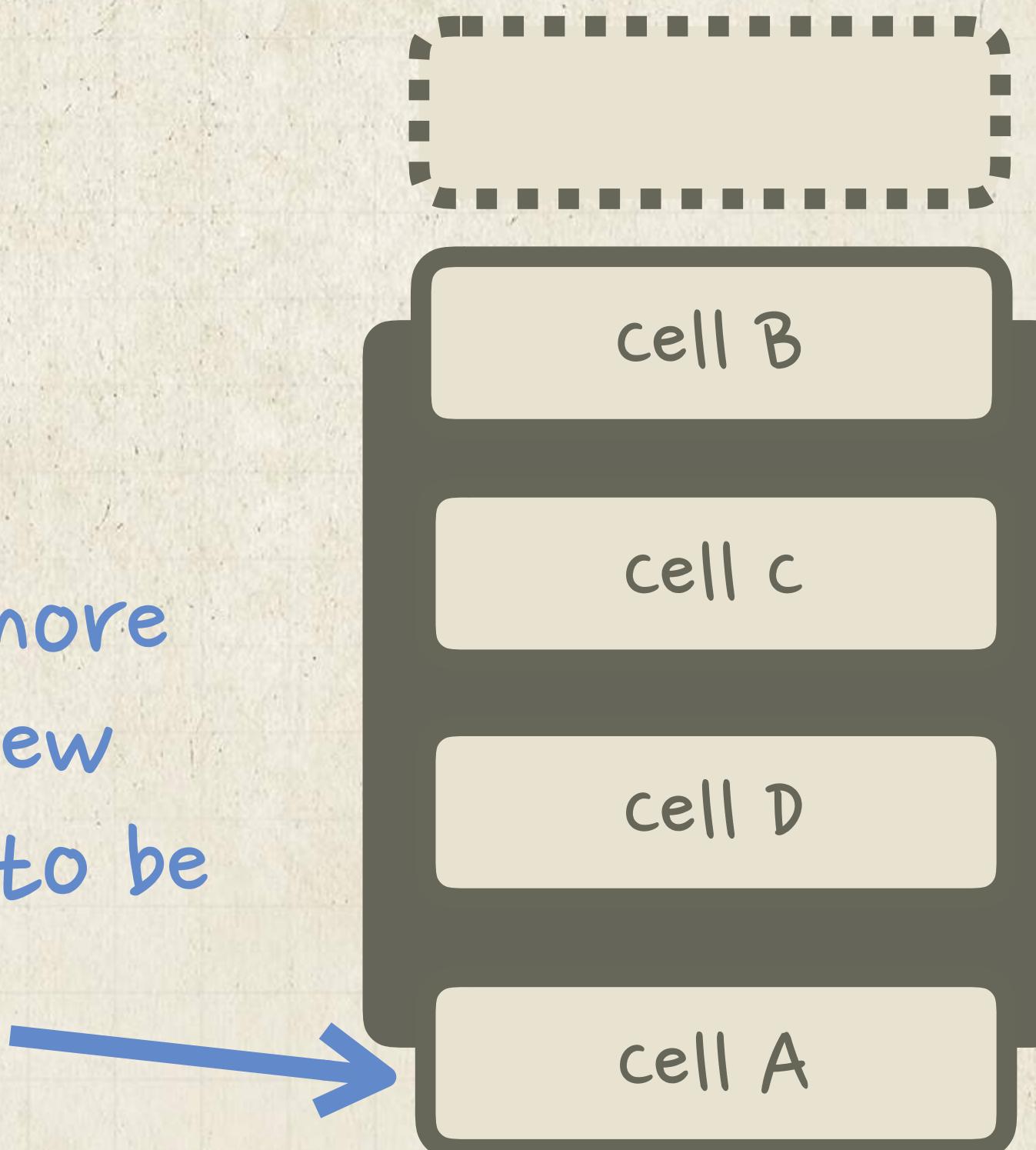


# How Table View Cell Recycling Works

Table cells are recycled as they scroll offscreen, and any new cells are just recycled cells filled in with new data.



After even more  
scrolling, a new  
cell is ready to be  
displayed



There's no need for a brand new cell because  
cell A would be recycled when it went offscreen

# Exploring the Table View Controller Swift File

## ProductsTableViewCellController.swift

```
import UIKit

class ProductsTableViewCellController: UITableViewController {

}
```



This is a view controller that has access to table-specific functions

The controller runs some of those functions automatically:

```
func tableView(_ tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int
```

Set the number of rows

```
func tableView(_ tableView: UITableView,
    cellForRowAt indexPath: IndexPath) -> UITableViewCell
```

create one cell for each row

# How to Read Function Declarations

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
```



The name of  
the function



These variables are  
available inside the body  
of the function



This function  
should return a  
number



# Using the Same Function Name Two Different Ways

Though the functions have the same name, since they have different parameters they are two different functions.

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
```

Different parameters



```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->  
UITableViewCell
```

# Setting the Number of Rows in the Table

## ProductsTableViewController.swift

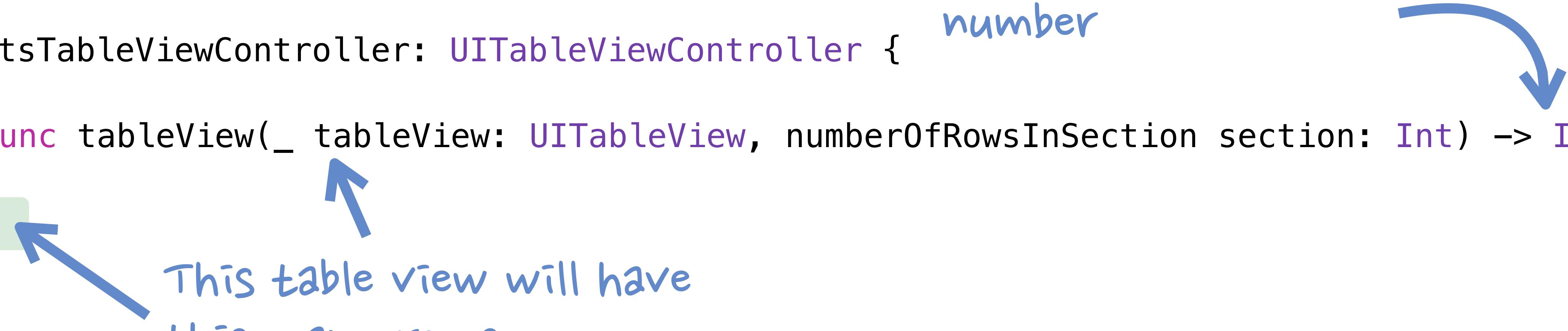
```
import UIKit

class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    {
        return 5
    }
}
```

This table view will have  
this many rows

This function should return a  
number



# Adding the Function That Will Create Cells

The `tableView(_ , cellForRowAt)` method is used to set up the cell that appears in each row

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    { ... }    Runs one time for each row (5 times in our case)

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
    }
}
```

A blue arrow points from the text "Runs one time for each row (5 times in our case)" down to the `cellForRowAt` function. Another blue arrow points from the text "This function should return a cell" back up to the `cellForRowAt` function.

This function should return a cell

# Creating a Cell for Each Row

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

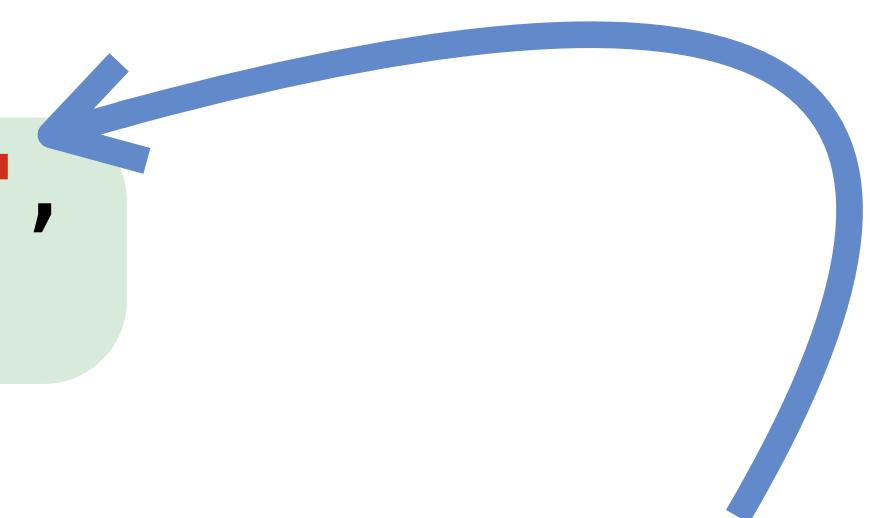
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    { ... }

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ProductCell",
            for: indexPath)

        return cell
    }
}
```

Return the created cell

This matches the identifier we added in the Storyboard

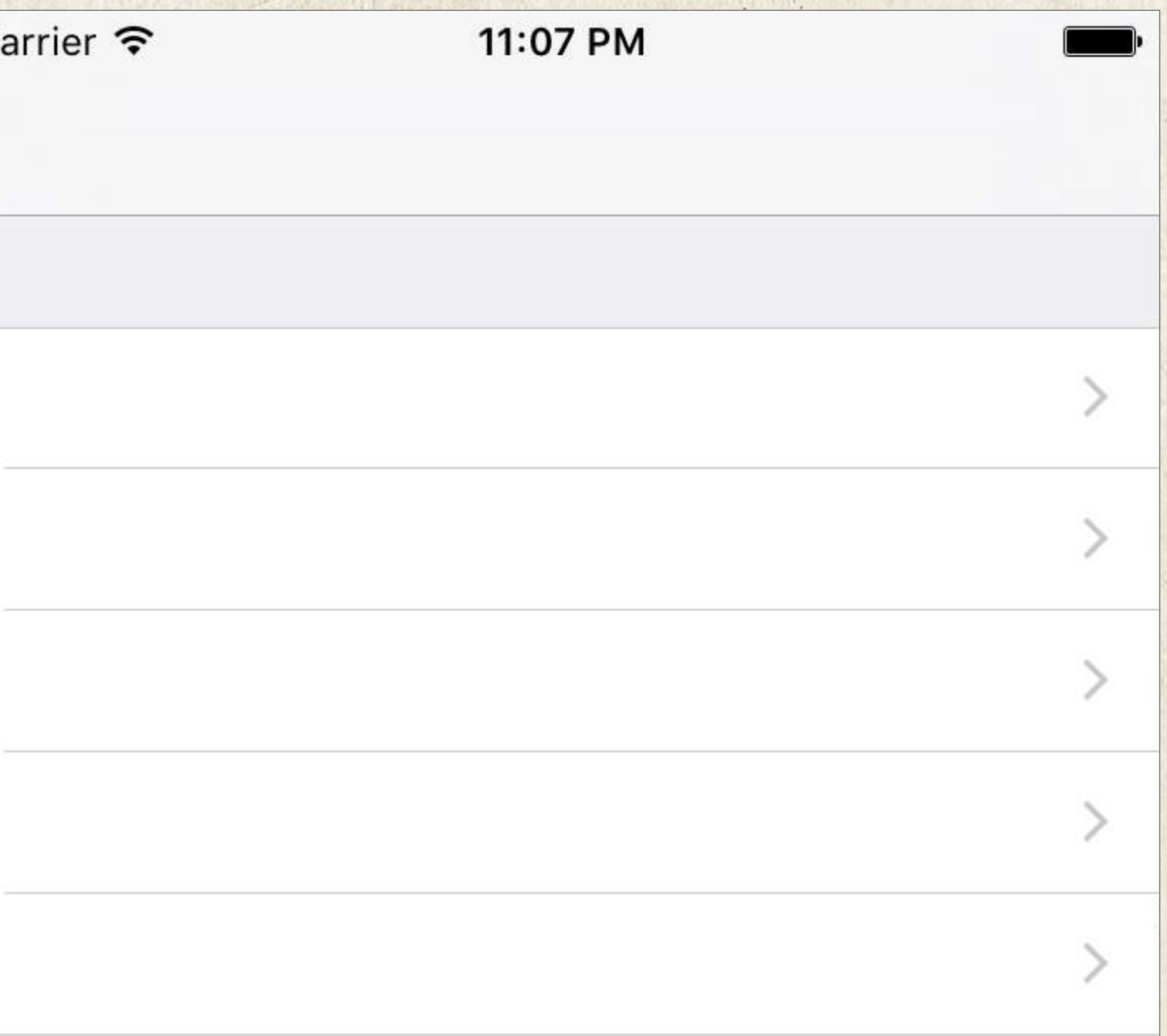


# Screencast: Switching to Grouped Cells

---



# Demo: Table View With Cells

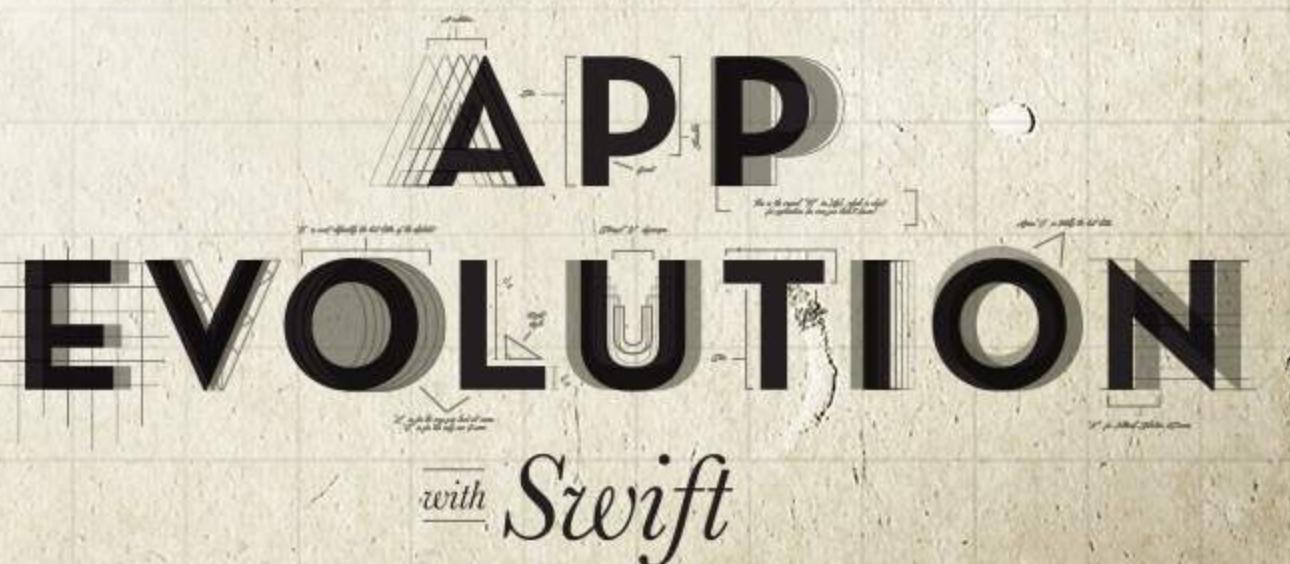


APP  
EVOLUTION  
*with Swift*

# Level 4

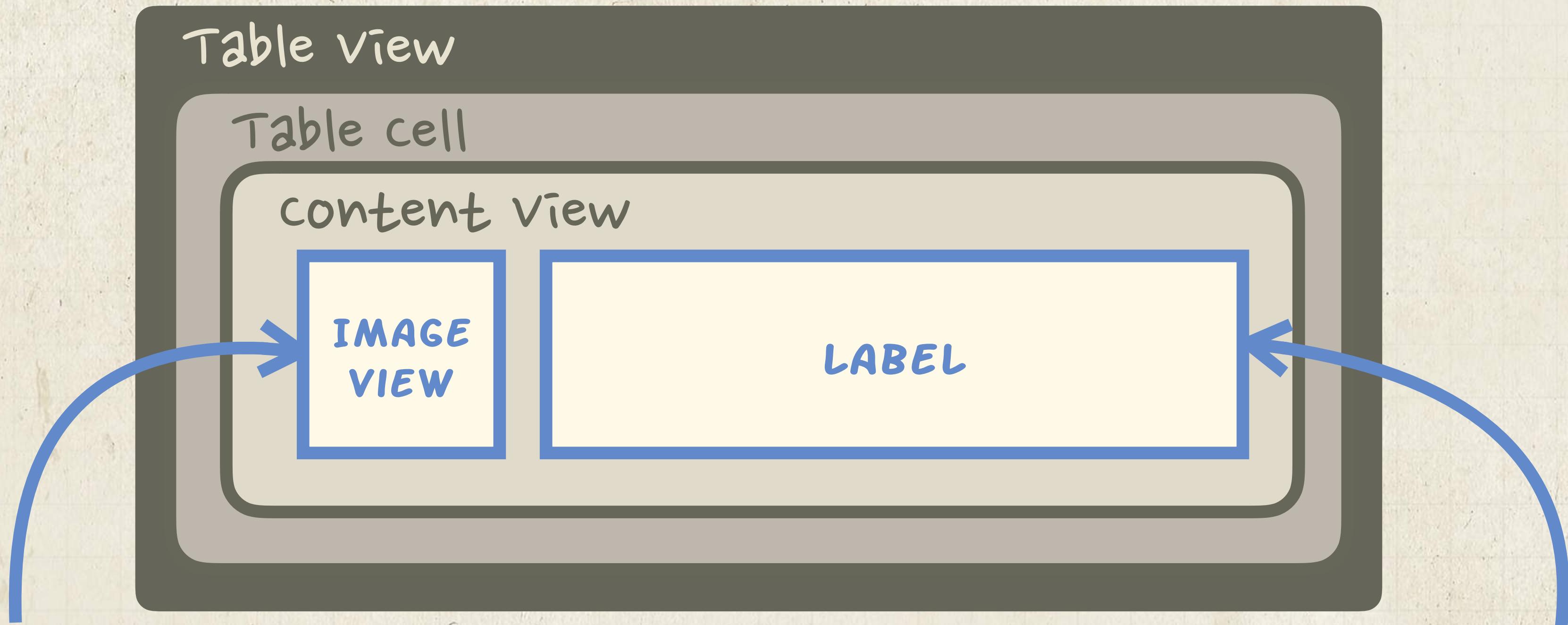
## Table Views

Section 2 - Displaying Data in Table Cells



# Table View Cells Come Pre-loaded With Subviews

Standard table cells come with an image view and label already wired up



This subview is already linked with  
the name `imageView`

This subview is already linked with  
the name `textLabel`

# Creating a Cell for Each Row

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ProductCell",
            for: indexPath)
        cell.textLabel.text = "Hello friend."
        return cell
    }
}
```

Let's see how we can fix this error



Value of optional type 'UILabel?' not unwrapped; did you mean to use '!' or '?'?

# A Quick Look Into the Table View Cell Docs

## Apple docs for UITableViewCell textLabel

Returns the label used for the main textual content of the table cell.

## Declaration

```
var textLabel: UILabel? { get }
```



This means it's optional.

It may or may not exist

Here's the official documentation for the cell's **textLabel**.



# First, Consider This Situation

It's possible to crash your app if you try to access an object or property that doesn't exist.

It's possible to create a custom cell without a `textLabel`

```
cell.textLabel.text = "Hello friend."
```

That would mean  
this property  
wouldn't exist...

which means  
this text  
property  
wouldn't  
exist...

So setting it will  
crash the app!



# Using a conditional to check for an optional value

One way to work with optionals is to use a conditional to first check if a value exists, and if it does, continue running code

```
if cell.textLabel != nil {  
    cell.textLabel!.text = "Hello friend."  
}
```

But there's a great shortcut...



# Using Optional Chaining

Optional chaining is a way to check optionals without writing conditionals

```
cell.textLabel?.text = "Hello friend."
```



Adding this question mark after optional properties means "First check if this exists"

textLabel exists

→ continue running the line of code

textLabel doesn't exist

→ stop running the line of code



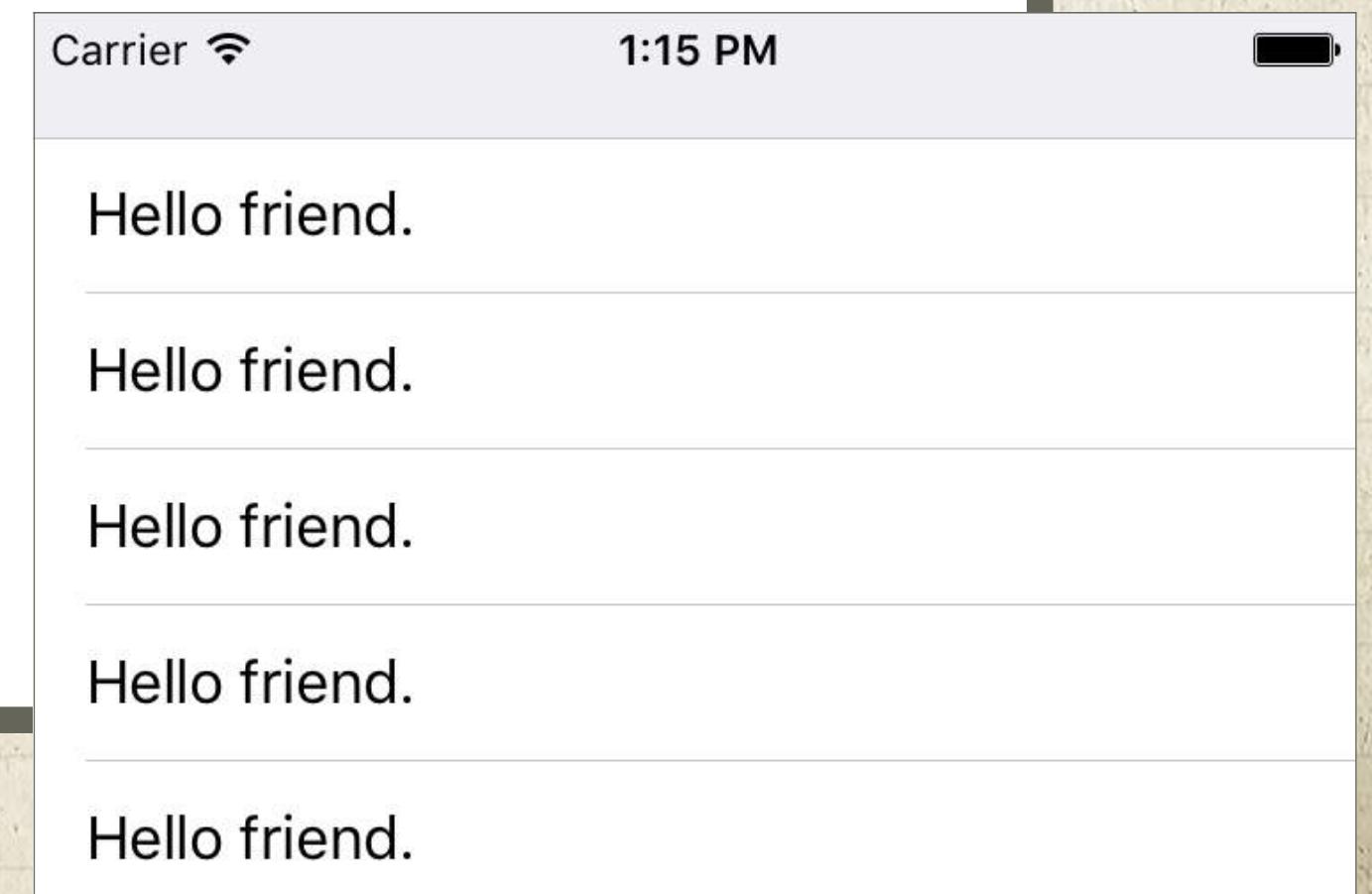
# Setting the `textLabel` of a Cell

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ProductCell",
                                                for: indexPath)
        cell.textLabel?.text = "Hello friend."
        return cell
    }
}
```



# Setting the imageView of a Cell

## ProductsTableViewController.swift

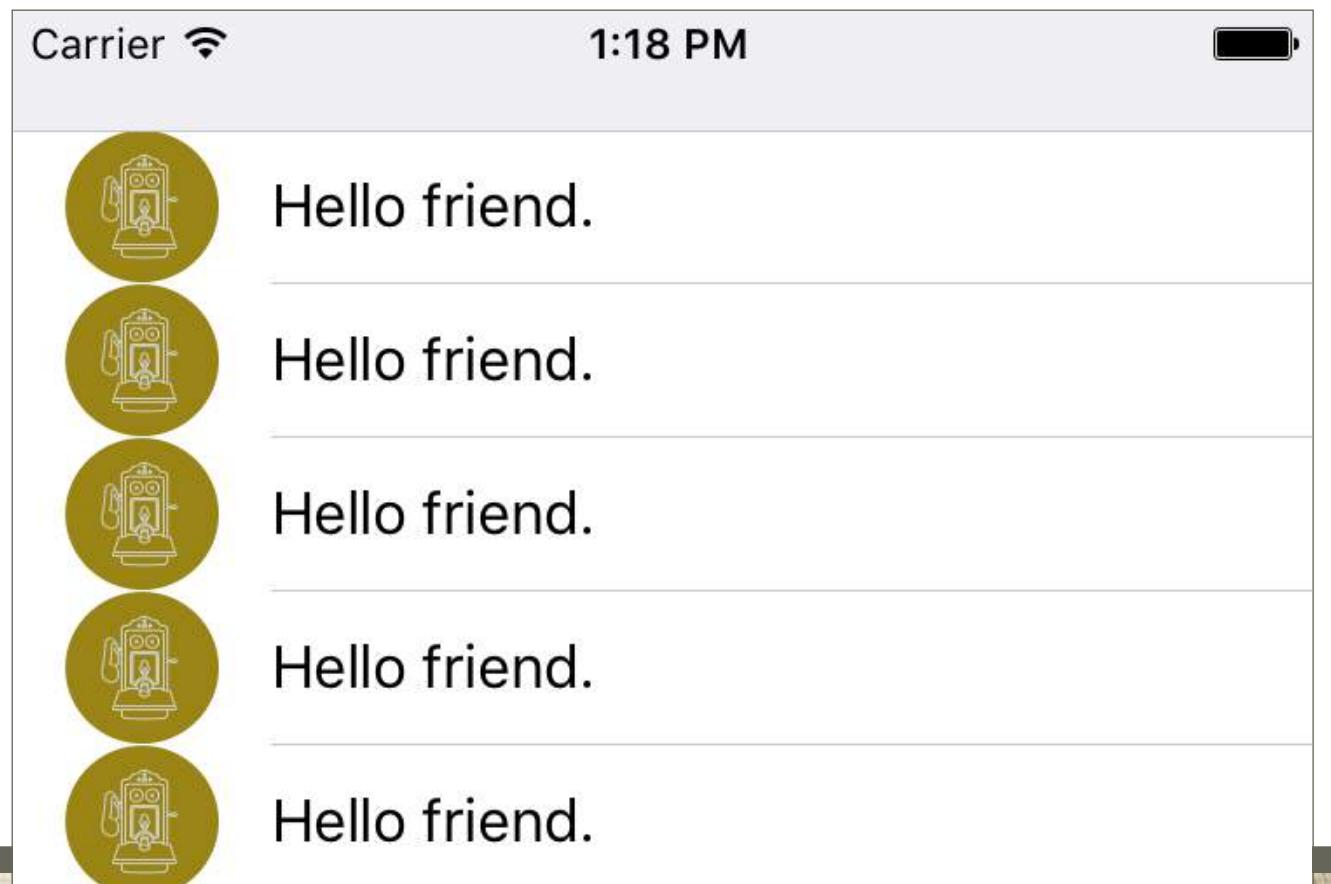
```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ProductCell",
                                                for: indexPath)
        cell.textLabel?.text = "Hello friend."
        cell.imageView?.image = UIImage(named: "image-cell1")

        return cell
    }
}
```

We've imported a few more images into the asset catalog



# Screencast: Adjusting the Table Cell Height

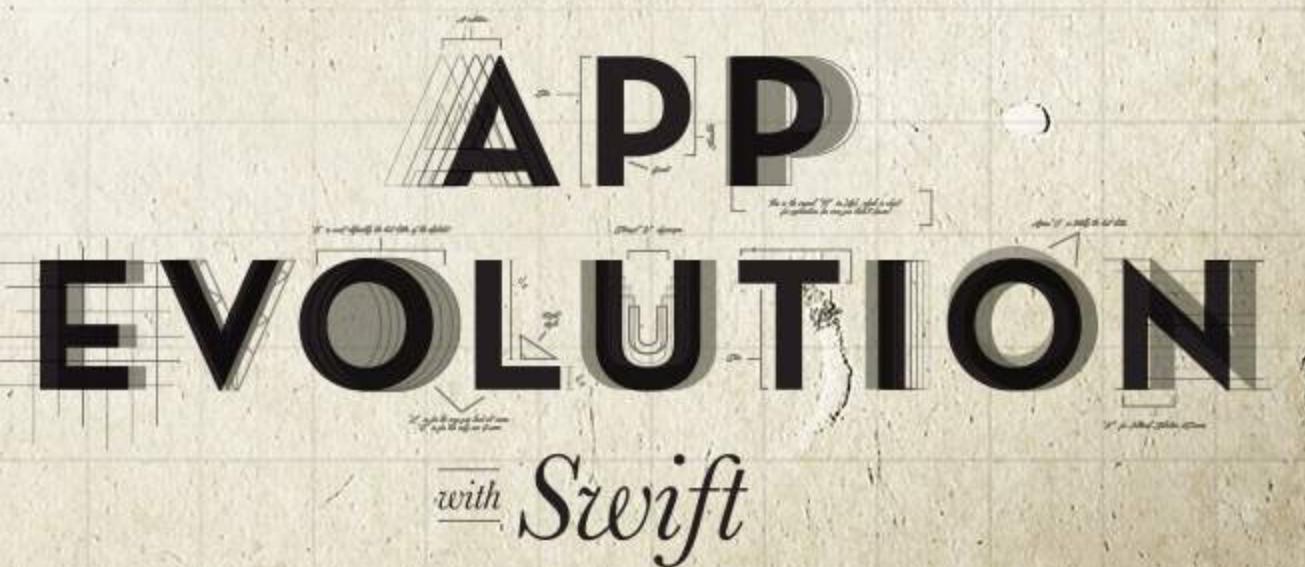
---



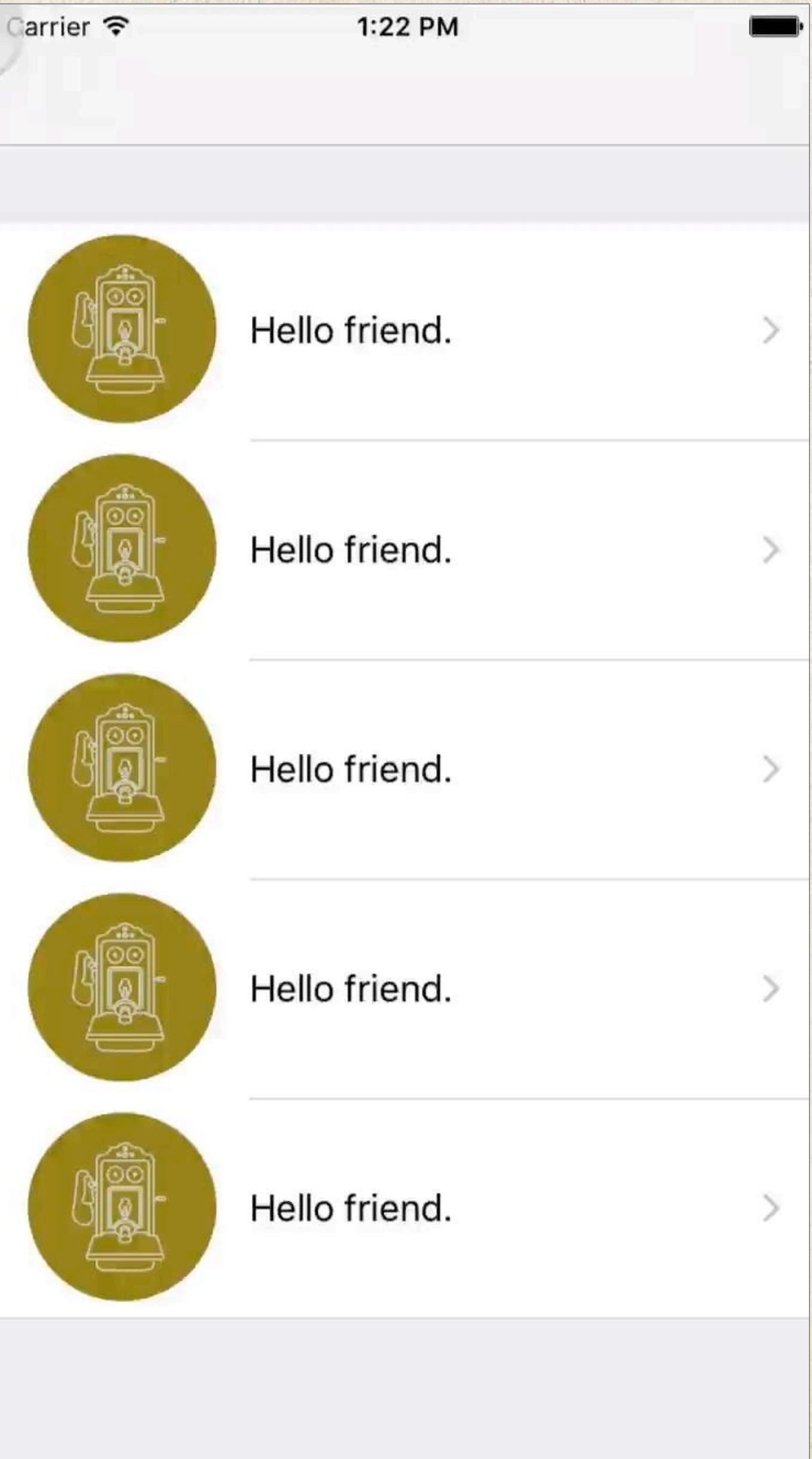
# Level 5

## Navigation

Section 1 - Transitioning Between View Controllers



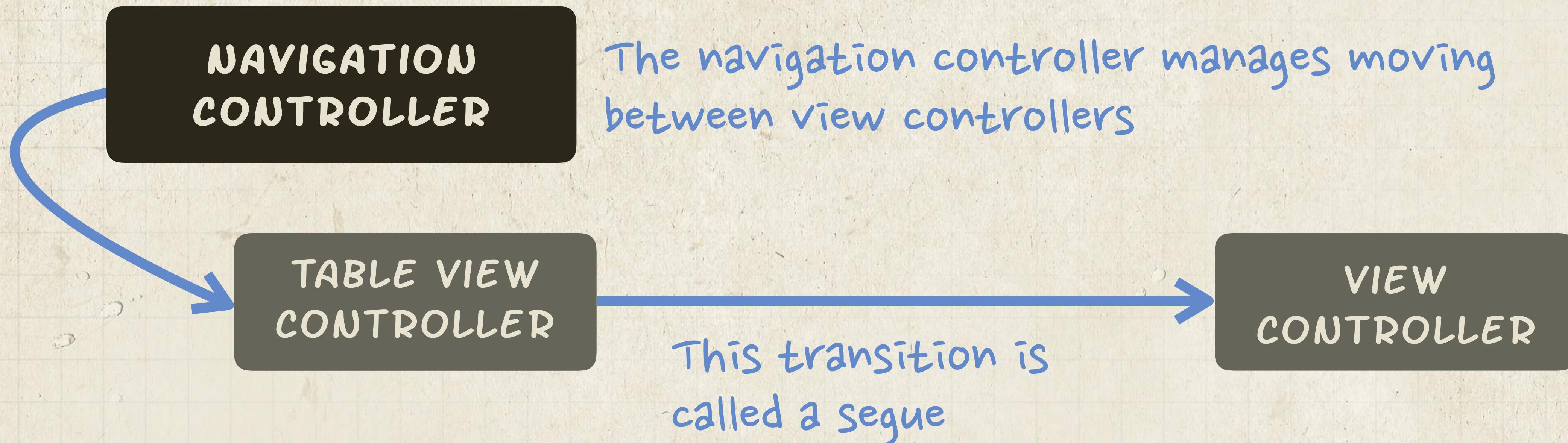
# Demo: Transitioning Between Screens



APP  
EVOLUTION  
*with Swift*

# Where Navigation Controllers Fit in the Hierarchy

You can't switch between view controllers unless they are a part of a navigation controller stack.



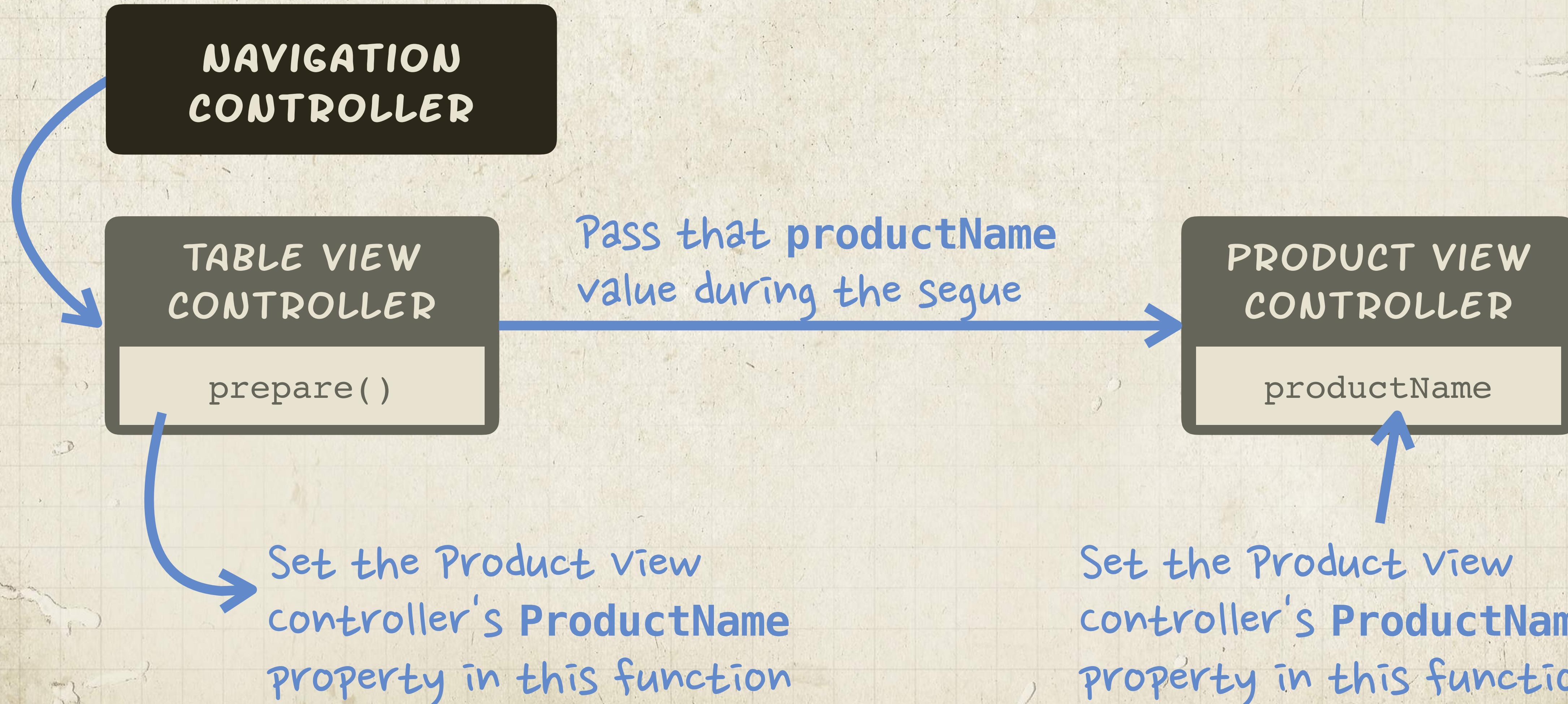
# Screencast: Adding a Navigation Controller

---



# Where Navigation Controllers Fit in the Hierarchy

You can access the segue object right before the segue happens and pass some data along with it.



# Start by Adding a Property to ProductViewController

## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

    ...
    var productName: String? ←
    override func viewDidLoad() {
        super.viewDidLoad()

    ...
    productNameLabel.text = "1937 Desk Phone"
}
}
```

Any data that doesn't exist until  
after the app starts has to be  
optional

# Assign the Passed-in Variable to the Label's Text

## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

    ...
    var productName: String?
    override func viewDidLoad() {
        super.viewDidLoad()
        ...
        productNameLabel.text = productName
    }
    ...
}
```

The value that's passed into  
this property will be displayed  
in the label

# Adding the prepare() Function

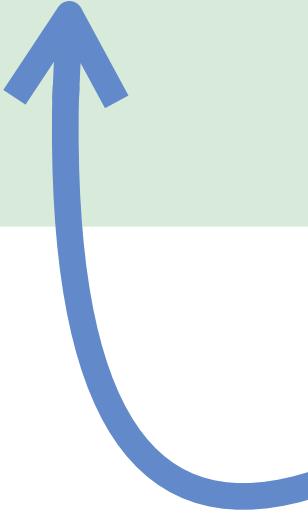
Add the `prepare()` function in the controller that you're coming *from*. In this case, that's the table view controller.

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

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



This runs every time a segue is triggered by an action

# Checking for the Right Segue

Since there can be multiple segues that all call the prepare() function, we first have to check the current segue's identifier.

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

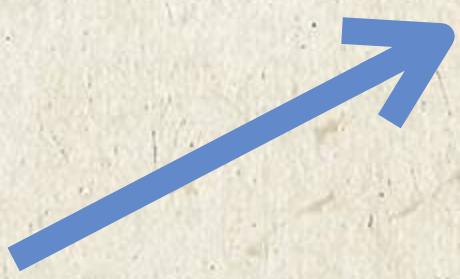
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "ShowProduct" {
            ...
        }
    }
}
```

Remember when we set this name  
in the storyboard?

# A Quick Look Into the Storyboard Segue Docs

The segue keeps a copy of the view controller it is transitioning *to*.

We can access the  
"to" view controller  
with this segue  
property



## Apple docs for UIStoryboardSegue destination

The destination view controller for the segue.

### Declaration

```
var destination: UIViewController { get }
```

PP  
UTION

with  
Swift

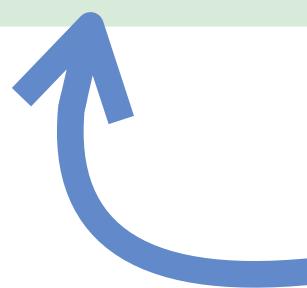
# Capturing the Destination in a Variable

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "ShowProduct" {
            let productVC = segue.destination
        }
    }
}
```



That's not enough — we need to say what kind of view controller this is

# Using the “as” Keyword to Change the Data Type

We can use “as” to convert from 1 data type to another.

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "ShowProduct" {
            let productVC = segue.destination as ProductViewController
        }
    }
}
```



Here, we're trying to let the compiler know that our destination view controller is a `ProductViewController`

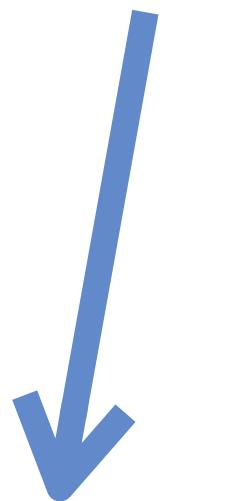
# Problem: Error When We Try to Set the Type

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "ShowProduct" {
            let productVC = segue.destination as ProductViewController
        }
    }
}
```



This code shows a compiler error



'UIViewController' is not convertible to 'ProductViewController'; did you mean to use 'as!' to force downcast?

If the compiler isn't sure, it won't let you compile code

If it guesses and is wrong, the app will crash!

# Using Downcasting to Suggest Object Types

The compiler doesn't know for sure if this is a  
**ProductViewController** object

```
let productVC = segue.destination as? ProductViewController
```



Optional now



Adding the question mark to "as" here returns an  
optional **ProductViewController**



# Set the Property on the ProductViewController

## ProductsTableViewController.swift

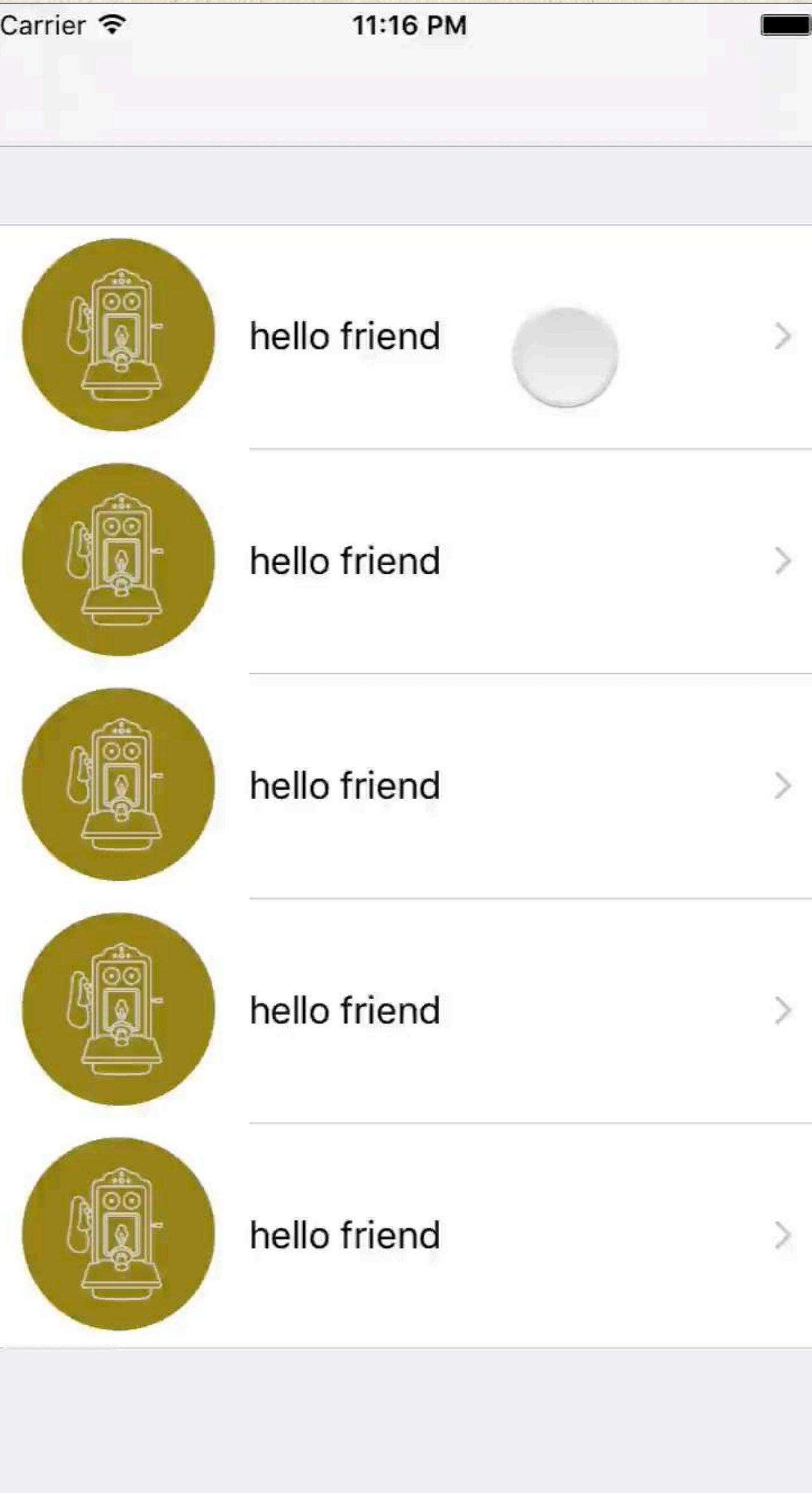
```
import UIKit

class ProductsTableViewController: UITableViewController {
    ...

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "ShowProduct" {
            let productVC = segue.destination as? ProductViewController
            productVC?.productName = "Really old phone"
        }
    }
}
```

Means "only set the name if productVC exists"

# Demo: Passing a Value to Another View Controller

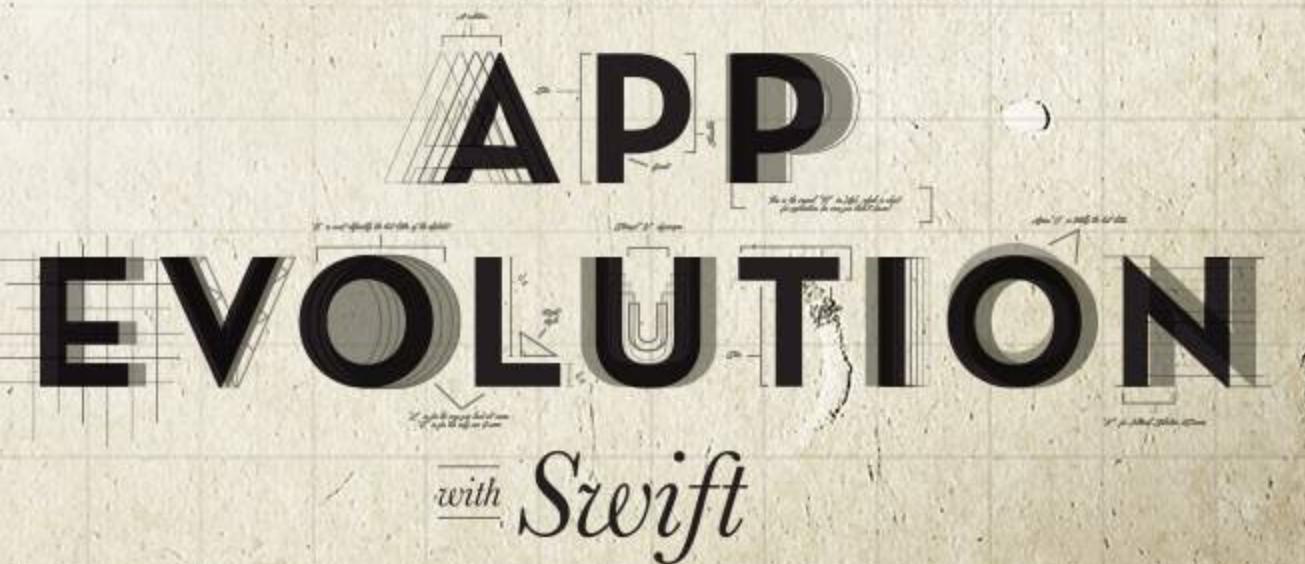


APP  
EVOLUTION  
*with Swift*

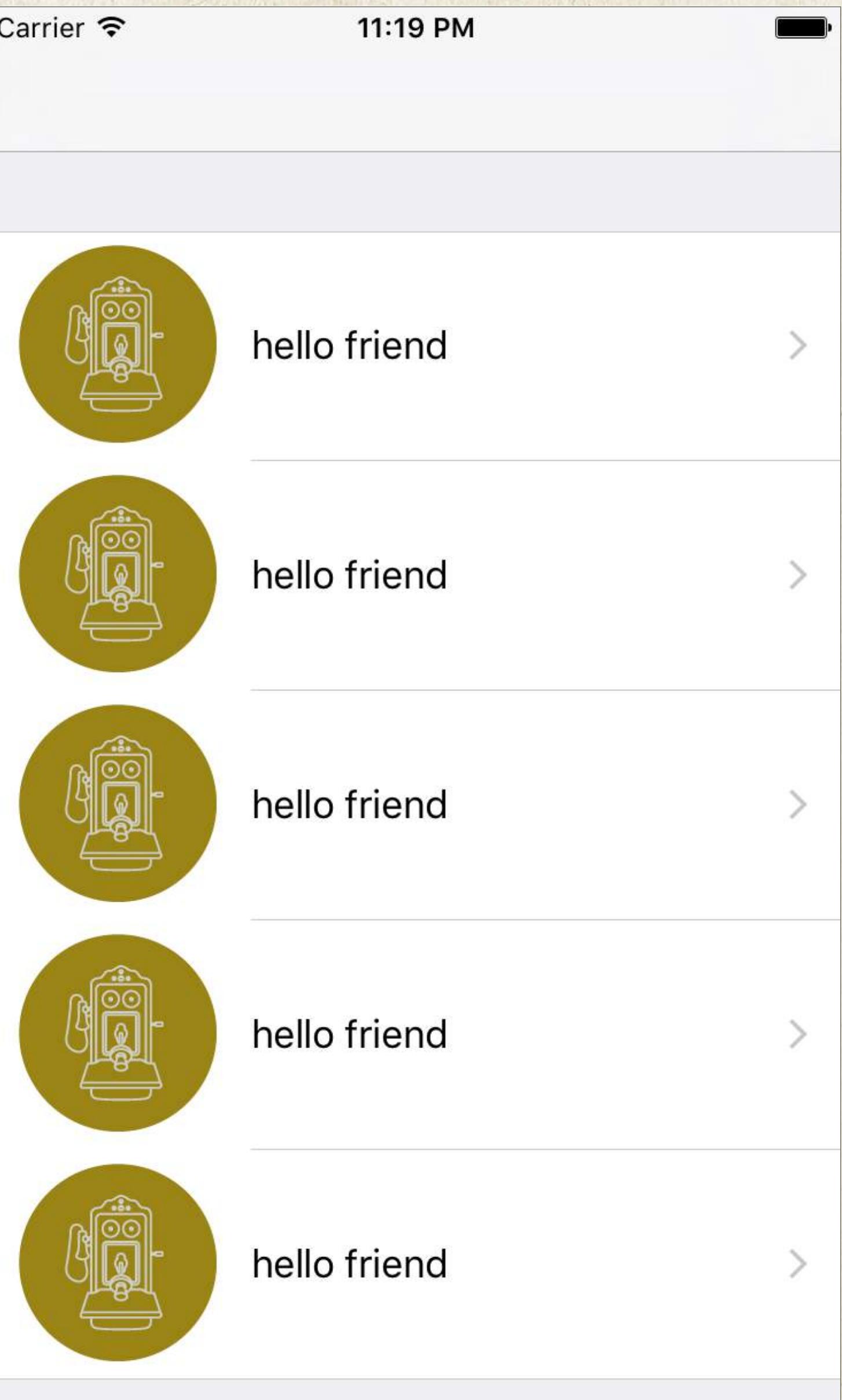
# Level 5

## Navigation

Section 2 - Displaying Dynamic Table Cell Data



# Problem: Cell Text Is Hard-coded



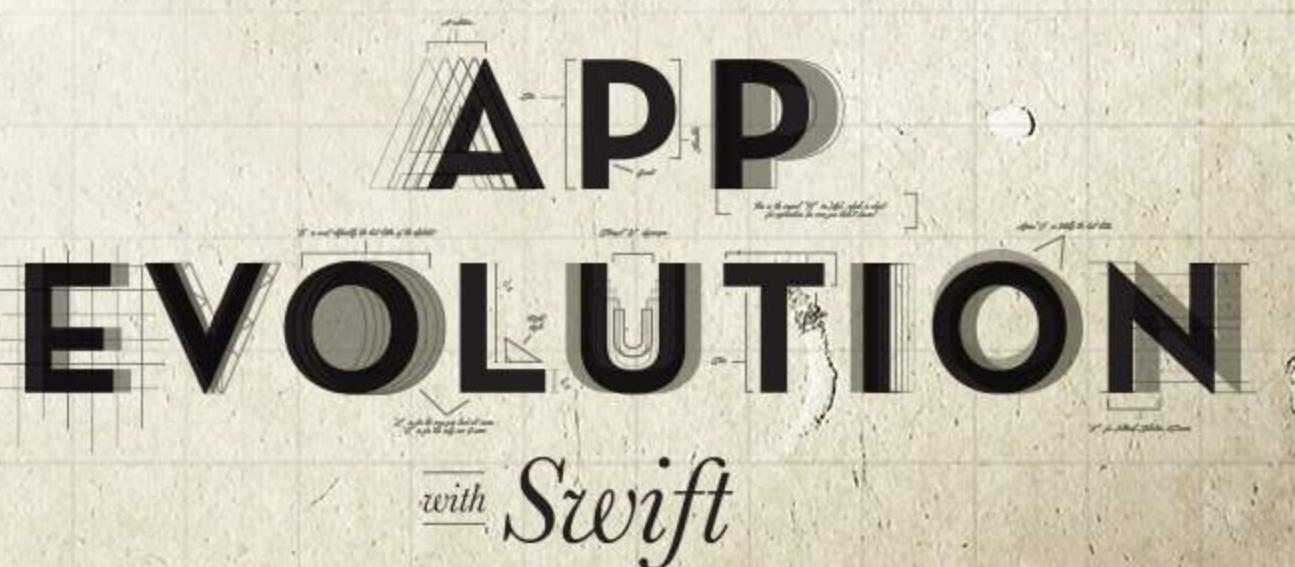
APP  
EVOLUTION  
*with Swift*

# Steps to Dynamic Data in Cells

---

## Steps in ProductsTableViewController

1. Store multiple names in an Array instead of a String
2. Update the Table View required methods to use data from that Array



# Arrays Can Store Multiple Values of a Single Type

## ProductsTableViewController.swift

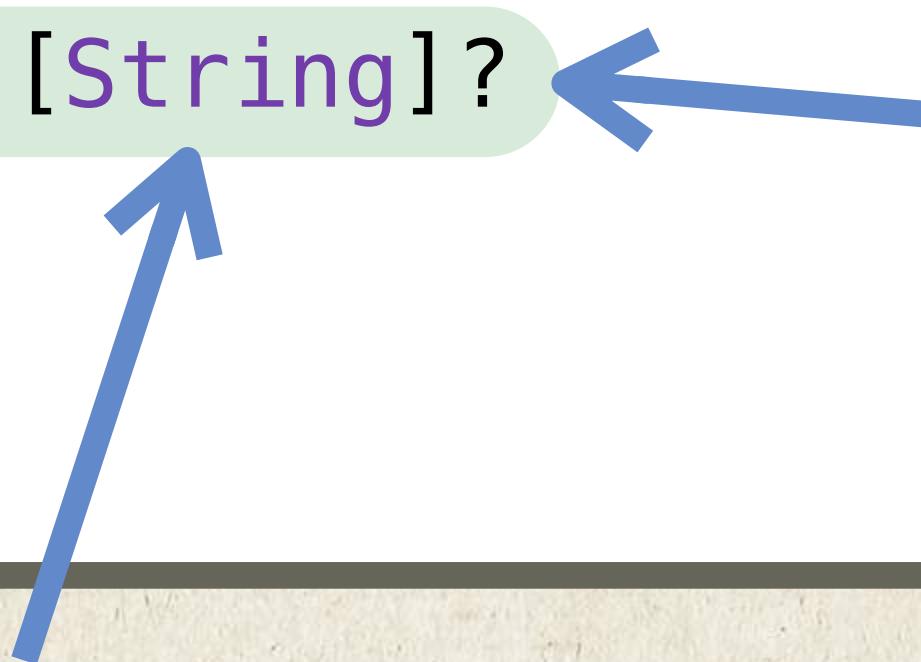
```
import UIKit

class ProductsTableViewController: UITableViewController {

    var productNames: [String]?

    ...
}
```

This array can only contain strings



This array is also optional

# Setting the Array Values in viewDidLoad

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

    var productNames: [String]?

    override func viewDidLoad() {
        super.viewDidLoad()

        productNames = ["1907 Wall Set", "1921 Dial Phone",
                        "1937 Desk Set", "1984 Motorola Portable"]

    }

    ...
}
```

We've set the array to have 4 values

# Plan of Attack for Updating the Cells

We have to update these 2 required table view functions to use data from the array instead of just hard-coded strings.

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    { ... }

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    { ... }
    ...
}
```

update this function to return the number  
of items in the array

update this function to use the one of the  
names in the array for each cell

# A Problem Updating the Number of Rows

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    {
        return 5
        return productNames?.count
    }

    override func tableView(_ tableView: UITableView,
                          cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        ...
    }
}
```

← count returns the number of items in the array

We get this error

Value of optional type 'Int?' not unwrapped; did you mean to use '!' or '?'?

# The Problem is We're Returning an Optional Int

## ProductsTableViewController.swift

```
import UIKit

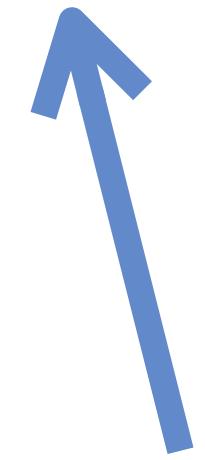
class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    {
        return 5
        return productNames?.count
    }

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    { ... }
    ...
}
```

If `productNames` doesn't exist,  
we'd be returning an optional int

This function wants us  
to return a regular Int,  
not an optional one



Value of optional type 'Int?' not unwrapped; did you mean to use '!' or '?'?

# Fixing the “Type not unwrapped” Error With if let

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    {
        return 5
        if let pNames = productNames {
            return pNames.count
        }
    }

    override func tableView(_ tableView: UITableView,
    cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        ...
    }
}
```

 A green circular icon with a white checkmark inside, positioned to the left of the code editor.

A blue callout bubble highlights the code block starting with `if let pNames = productNames {`. An arrow points from the text "Get the count of the non-optional version of the array" to the `pNames.count` part of the code. Another arrow points from the text "If the array exists, create a non-optional version of it" to the `if let` keyword.

If the array exists, create a non-optional version of it

Get the **count** of the non-optional version of the array

# Return Zero Rows If the Array Doesn't Exist

## ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

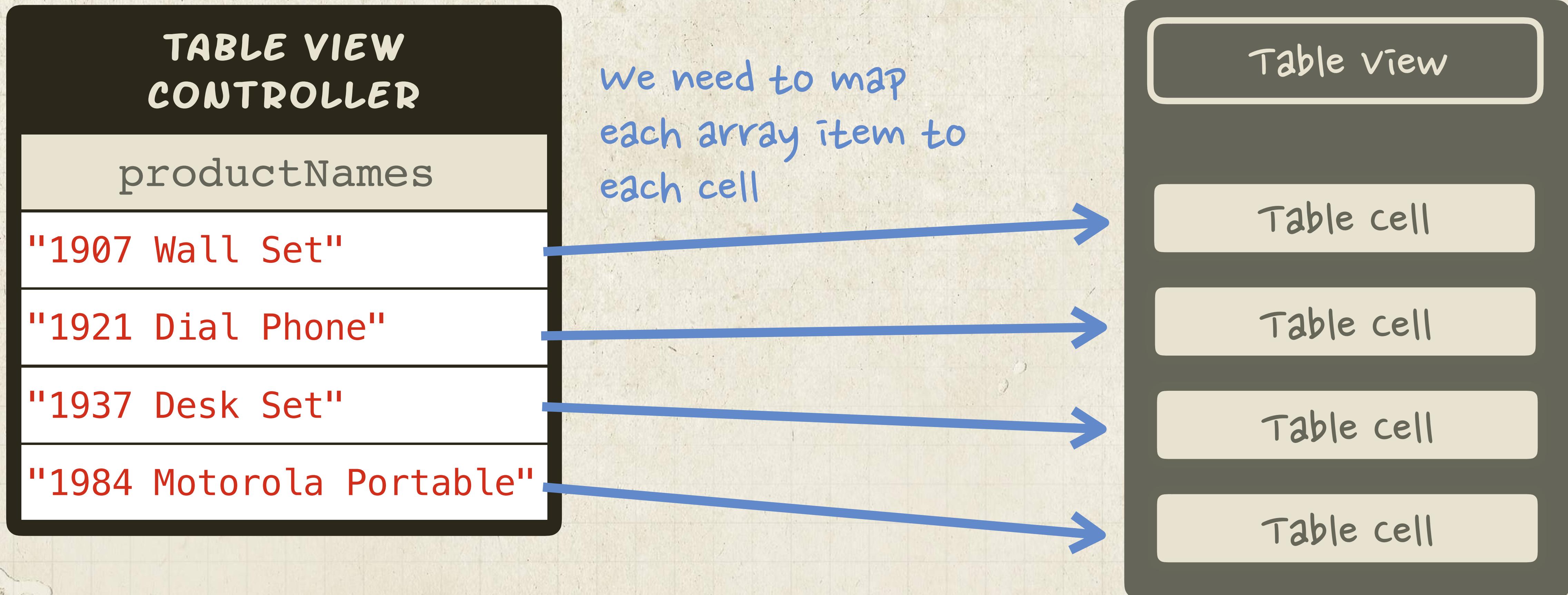
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 5
        if let pNames = productNames {
            return pNames.count
        }
        return 0
    }

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        ...
    }
}
```

If `productNames` doesn't exist, we still need to return zero rows

# Mapping Array Values to Table Cells

Each item in the array will correspond to each row in the table.



# Unwrap the Optional productName Variable

## ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    { ... }

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ProductCell", for: indexPath)

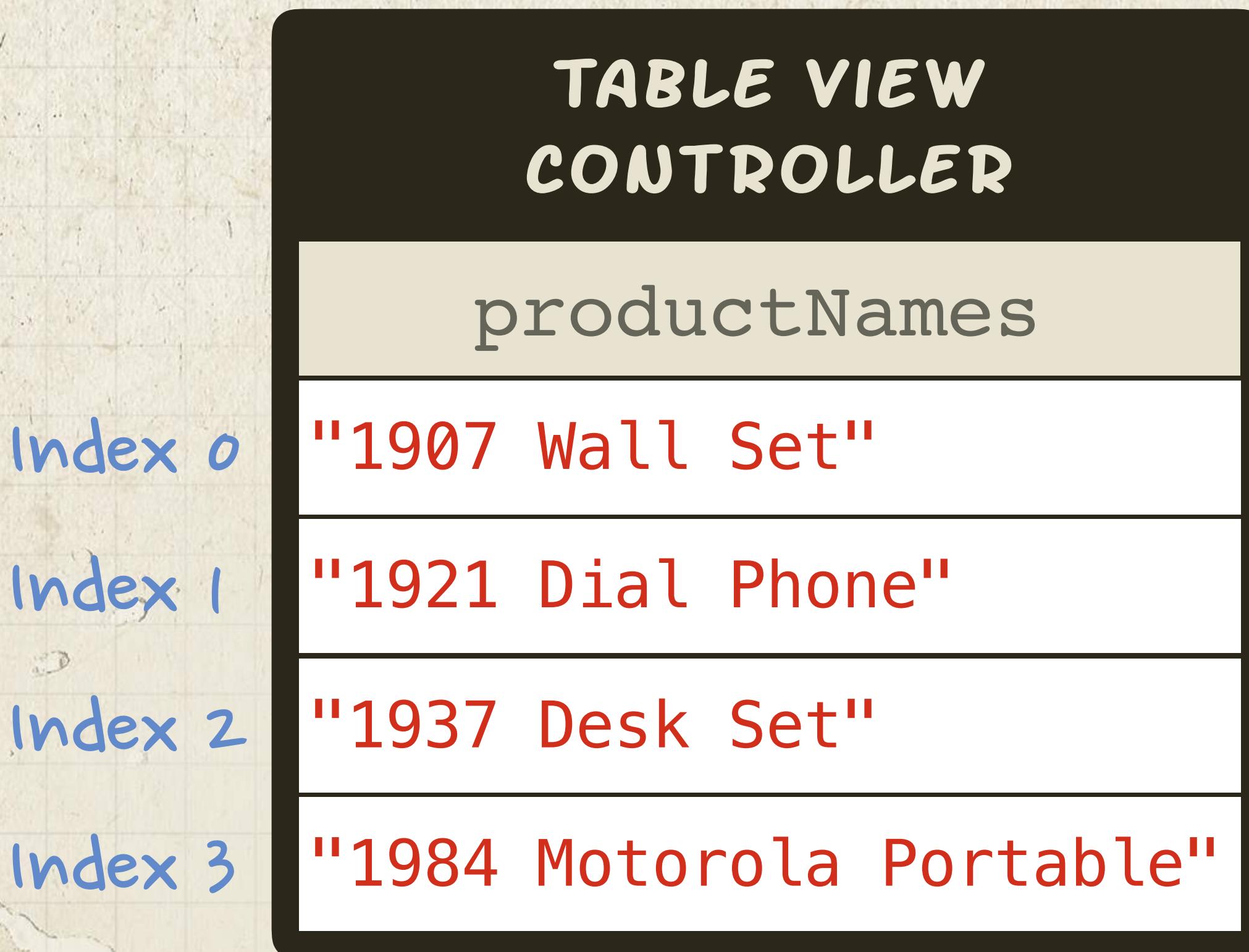
        let productName = // set the product name

        if let pName = productName {
            cell.textLabel?.text = pName
        }
        return cell
    }
    ...
}
```

The product name will be an optional, so we need to unwrap it before setting the label to that text

# Reading Values From an Array

Each value in the array can be accessed by typing a number between square brackets after the variable name.

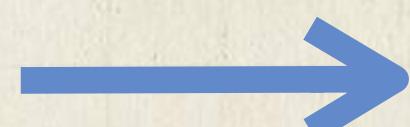


```
print(productNames[0])
```



"1907 Wall Set"

```
print(productNames[3])
```



"1984 Motorola Portable"

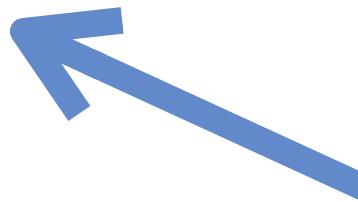
# Plan of Attack for Updating the Cells

## ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {

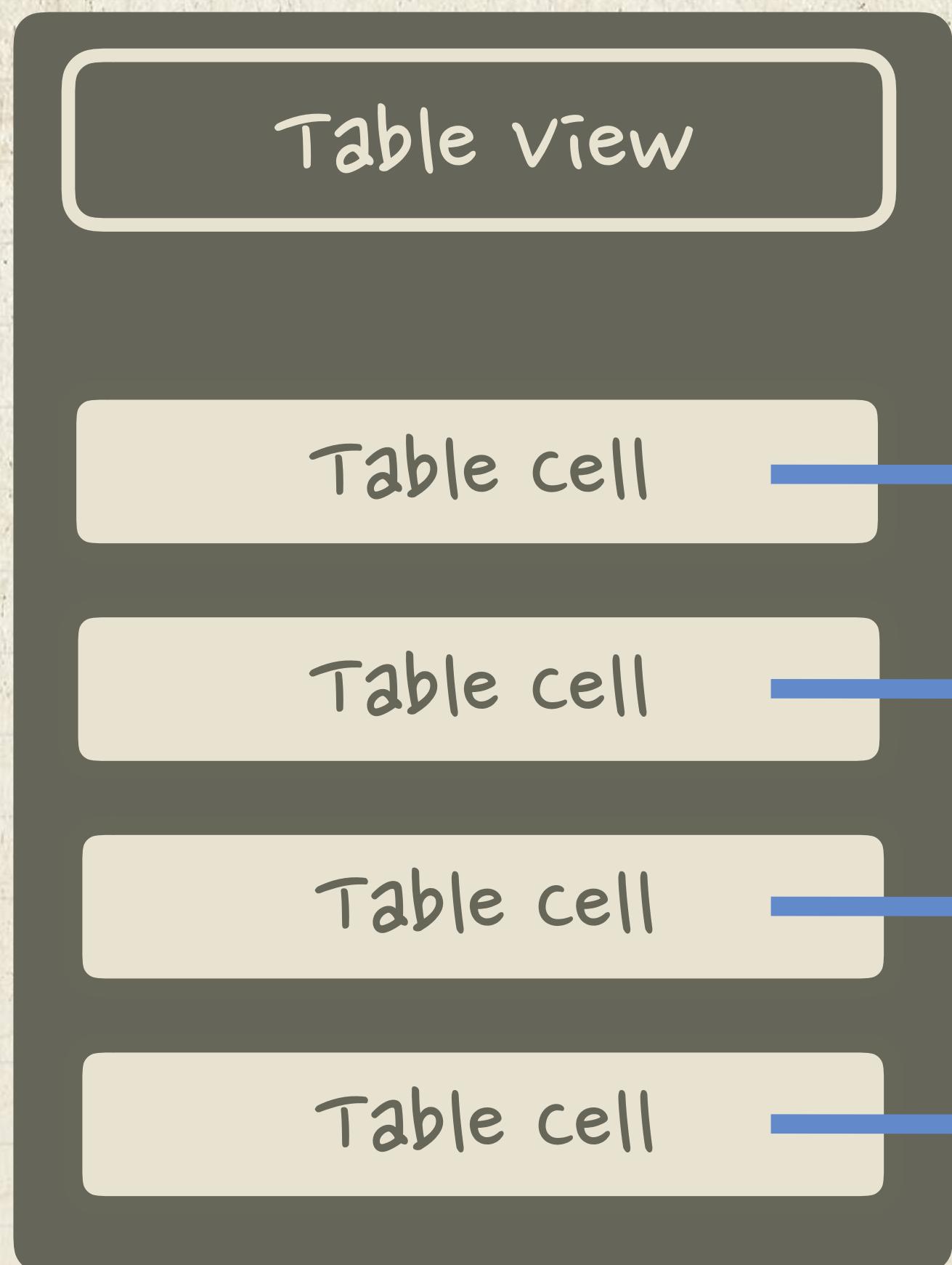
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    { ... }

    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ProductCell", for: indexPath)

        let productName = productNames[0] productNames[0] 
        if let pName = productName {
            cell.textLabel?.text = pName
        }
        return cell
    }
    ...
}
```

This will get the first item in the array every time, but we want this number to be different for each cell

# Finding a Cell With an Index Path



Every cell has an  
index path

Every index path has a  
property called row  
that's just a number

Table cell

IndexPath

row

0

Table cell

IndexPath

row

1

Table cell

IndexPath

row

2

Table cell

IndexPath

row

3

Rows start at zero, just  
like array values!

# Using the indexPath to Access an Array Item

## ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    { ... }

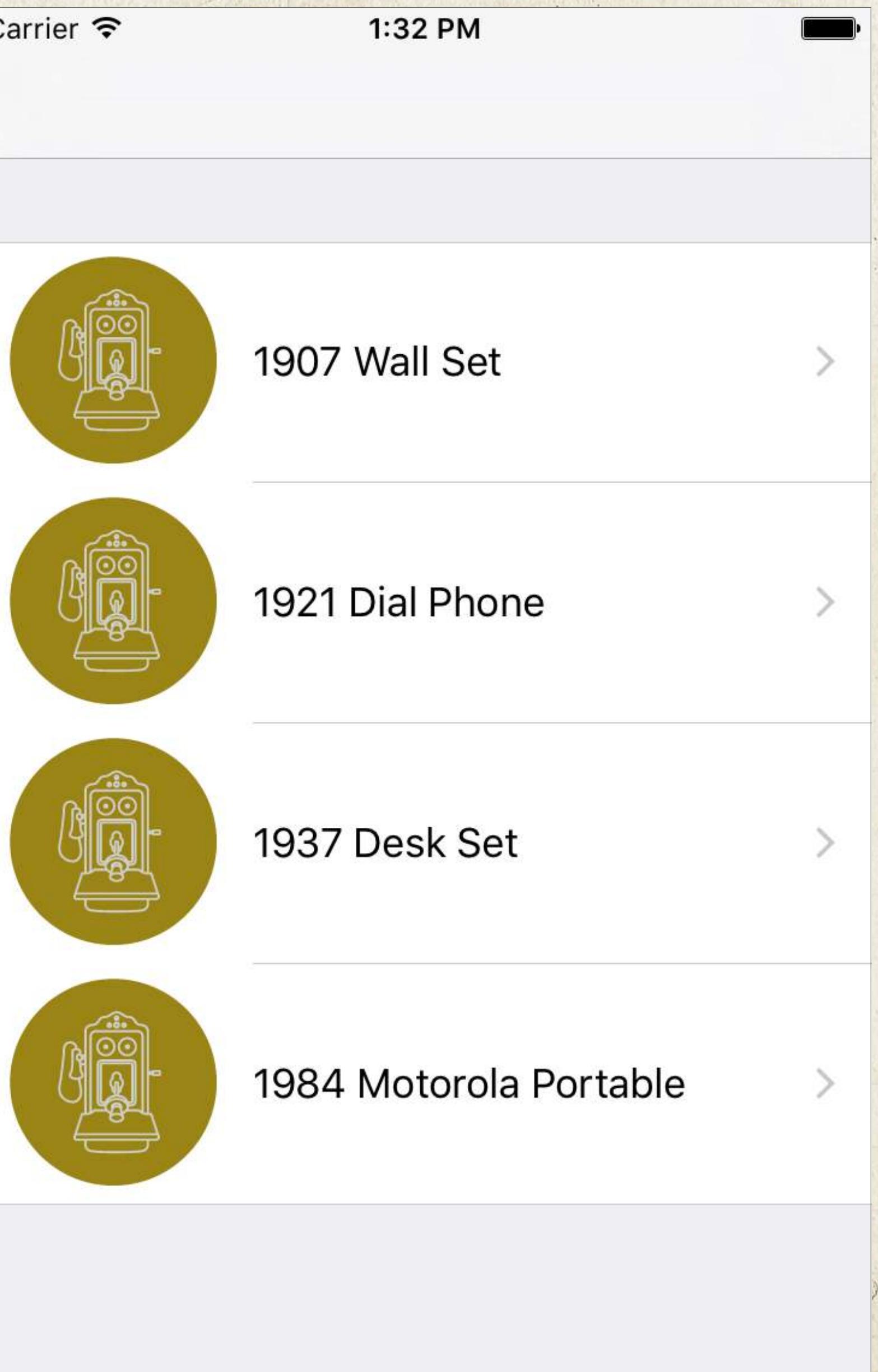
    override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ProductCell", for: indexPath)

        let productName = productNames?[indexPath.row]
  

        if let pName = productName {
            cell.textLabel?.text = pName
        }
        return cell
    }
    ...
}
```

Now we've got a copy of the name that  
should be displayed in this cell

# Demo: Dynamic Cell Text

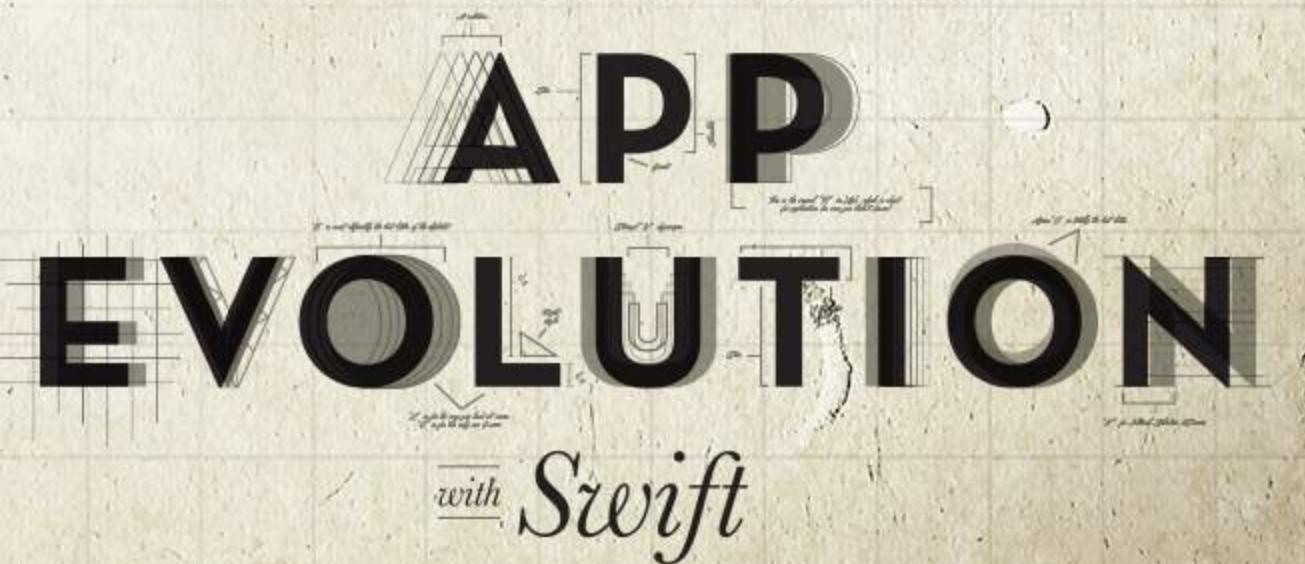


APP  
EVOLUTION  
*with Swift*

# Level 5

## Navigation

Section 3 - Passing Dynamic Data During a Transition



# Problem: We Want the Product Detail to Update Too



APP  
EVOLUTION  
*with Swift*

# The Plan for Updating prepare()

## ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {  
    ...  
  
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
        if segue.identifier == "ShowProduct" {  
            let productVC = segue.destination as? ProductViewController  
            productVC?.productName = "Really old phone" ←  
            // get the cell that was tapped  
            // get the index path for that cell  
            // use the index path to get the productName from the array  
            // send the product name to the product view controller  
        }  
    }  
}
```

Stop hard-coding the  
product name

# Getting a Copy of the Cell That Was Tapped

## ProductsTableViewController.swift

```
...  
    "Sender" Here is the cell that caused the segue to happen  
  
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController  
  
        let cell = sender as? UITableViewCell  
    }  
}  
}
```

Let the compiler know the  
sender is a UITableViewCells

# Using the Cell to Get an indexPath

## ProductsTableViewController.swift

...

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController  
  
        let cell = sender as? UITableViewCell  
        if let c = cell {  
            let indexPath = tableView.indexPath(for: c)  
        }  
    }  
}
```

This function returns an index path if you give it a cell

We have to unwrap first because cell is optional

# Using the indexPath to Get a Product Name

## ProductsTableViewController.swift

...

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController  
  
        let cell = sender as? UITableViewCell  
        if let c = cell {  
            let indexPath = tableView.indexPath(for: c)  
            if let ip = indexPath {  
                let productName = productNames?[ip.row]  
            }  
        }  
    }  
}
```

The indexPath that's returned is  
also an optional

use the indexPath to get the  
correct item from the array

# Send the Name to the ProductViewController

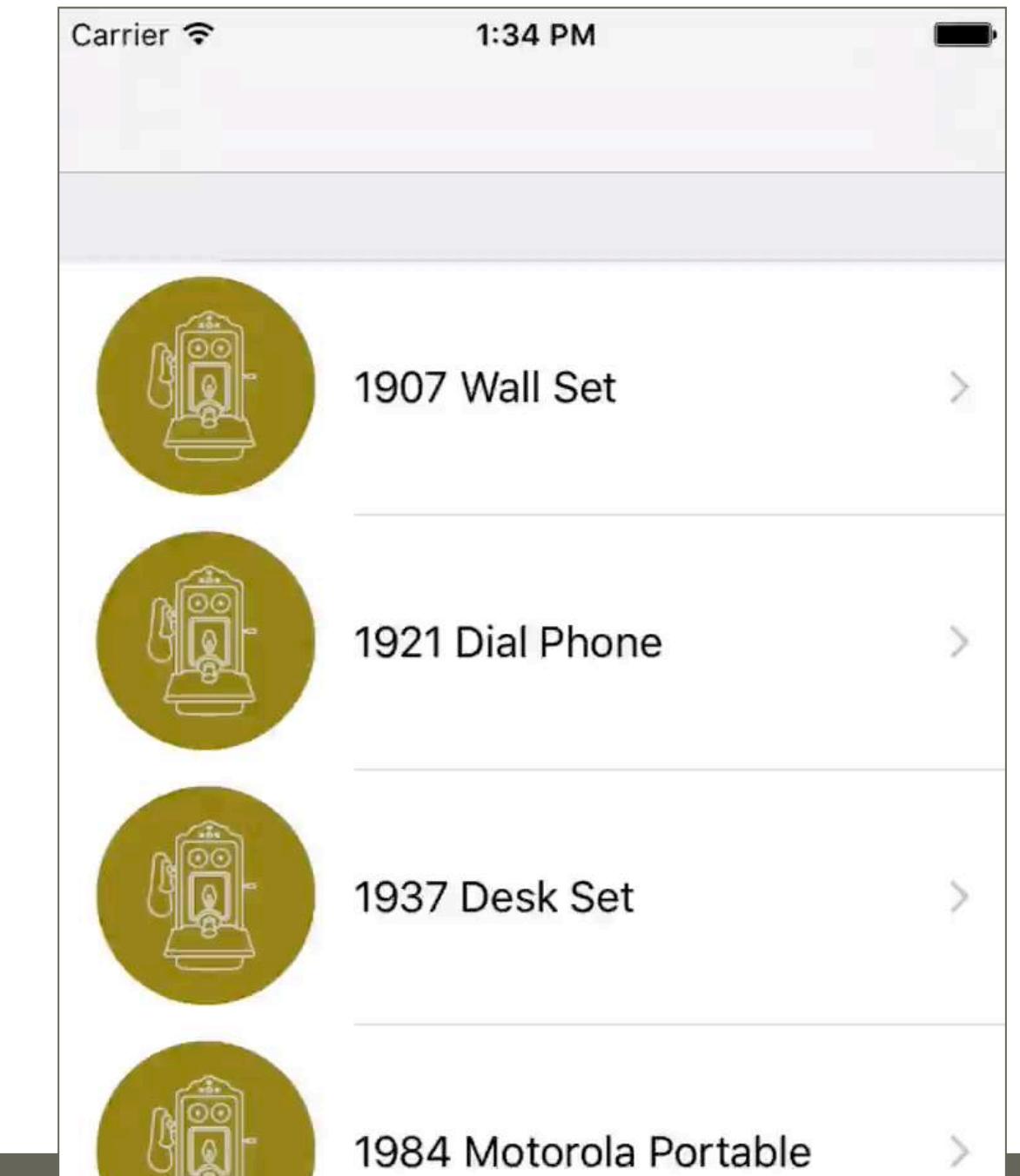
## ProductsTableViewController.swift

...

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController  
  
        let cell = sender as? UITableViewCell  
        if let c = cell {  
            let indexPath = tableView indexPath(for: c)  
            if let ip = indexPath {  
                let productName = productNames?[ip.row]  
                productVC?.productName = productName  
            }  
        }  
    }  
}
```



Set the **productName** for the  
product view controller



# Problem: This Is a Lot of Messy Nested Code

## ProductsTableViewController.swift



```
...  
    let cell = sender as? UITableViewCell  
    if let c = cell {  
        let indexPath = tableView.indexPath(for: c)  
        if let ip = indexPath {  
            let productName = productNames?[ip.row]  
            productVC?.productName = productName  
        }  
    }  
}
```

Lots of code block nesting

Ambiguous variable names

# Solution 1: Reuse Variable Names in if let

It's fine to use the same name to unwrap the optional variable.

## ProductsTableViewController.swift

```
...
let cell = sender as? UITableViewCell
if let cell = cell {
    let indexPath = tableView indexPath(for: cell)
    if let indexPath = indexPath {
        let productName = productNames?[indexPath.row]
        productVC?.productName = productName
    }
}
}
```

These are the old optionals

These are the new unwrapped values

# Solution 2: Unwrap and Set on the Same Line

## ProductsTableViewController.swift

...

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController
```

```
        if let cell = sender as? UITableViewCell {  
            let indexPath = tableView.indexPath(for: cell)  
            if let indexPath = indexPath {  
                let productName = productNames?[indexPath.row]  
                productVC?.productName = productName  
            }  
        }  
    }  
}
```

Get a copy of the cell and  
unwrap it in the same line

Before

```
let cell = sender as? UITableViewCell  
if let cell = cell {  
    ...  
}
```

# Refactor the indexPath Unwrapping to 1 Line

## ProductsTableViewController.swift

...

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController  
  
        if let cell = sender as? UITableViewCell {  
            if let indexPath = tableView.indexPath(for: cell) {  
                let productName = productNames?[indexPath.row]  
                productVC?.productName = productName  
            }  
        }  
    }  
}
```

Get a copy of the indexPath and unwrap it in the same line

Before

```
let indexPath = tableView.indexPath(for: cell)  
if let indexPath = indexPath {  
    ...  
}
```

# Refactor Setting the Product Name to 1 Line

## ProductsTableViewController.swift

...

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController  
  
        if let cell = sender as? UITableViewCell {  
            if let indexPath = tableView.indexPath(for: cell) {  
                productVC?.productName = productNames?[indexPath.row]  
            }  
        }  
    }  
}
```

Look up the right product name and assign it in one line

Before

```
let productName = productNames?[indexPath.row]  
productVC?.productName = productName
```

# Using Guard to Clean Up if let Nesting

## ProductsTableViewController.swift



...

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "ShowProduct" {  
        let productVC = segue.destination as? ProductViewController
```

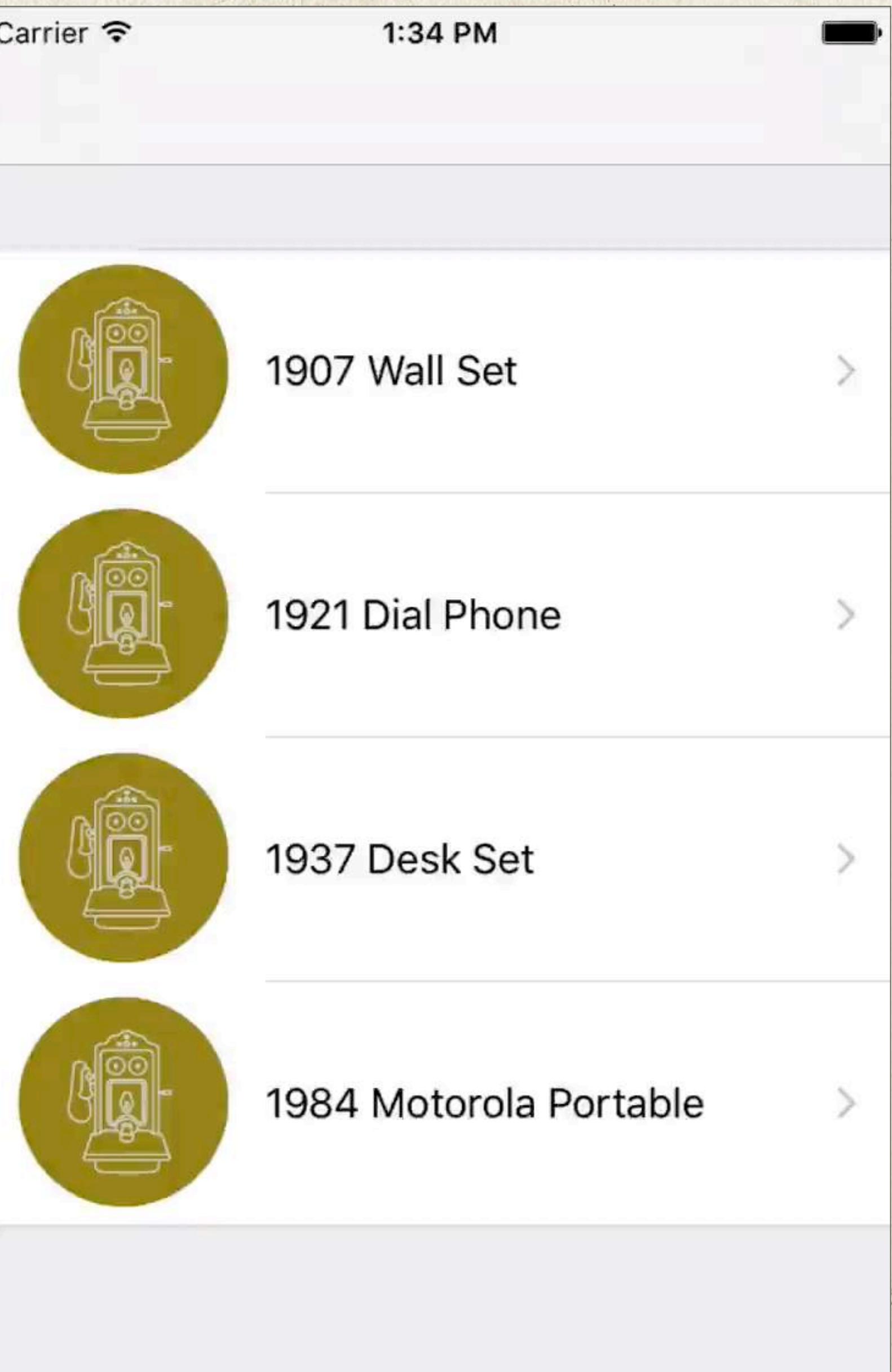
```
    guard let cell = sender as? UITableViewCell,  
          let indexPath = tableView.indexPath(for: cell) else {  
        return  
    }  
    productVC?.productName = productNames?[indexPath.row]  
}
```

This first checks if these values exist, and if they do it sets them to the **cell** and **indexPath** variables

stops running the function

Now safely use **indexPath**

# Demo: Segue Passing Over Product Data



APP  
EVOLUTION  
*with Swift*

# Level 6

## Custom Classes for Data



# A Problem With Our Current Approach

## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {
    ...
    var productName: String?
    ...
    override func viewDidLoad() {
        super.viewDidLoad()
        ...
        productNameLabel.text = productName
    }
}
```

we can only pass one piece of data

What if we wanted to pass the product and cell images?

# One Option for Dealing With Passing Multiple Values

## ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

    ...

    var productName: String?
    var cellImageName: String?
    var productImageName: String?

    override func viewDidLoad() {
        super.viewDidLoad()

        ...
        productNameLabel.text = productName
    }
}
```



We could create  
multiple variables, but  
there's a better way!



# Screencast: Creating a New Class for Data

---

