
APP
EVOLUTION
with Swift

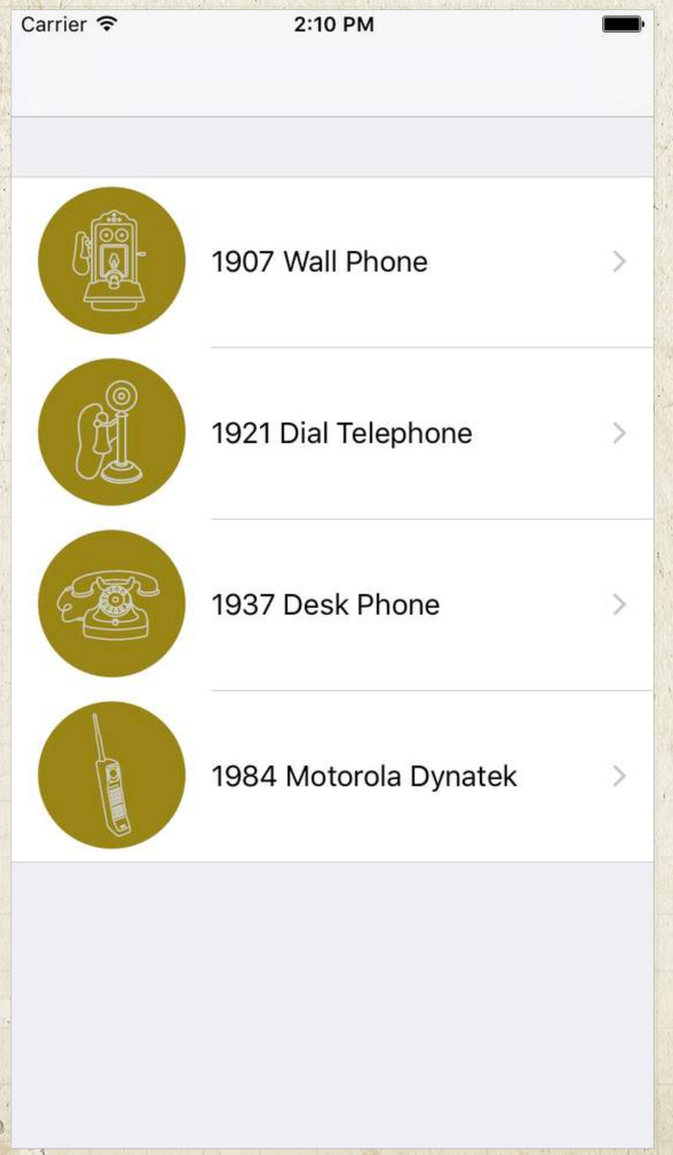
What is Swift?

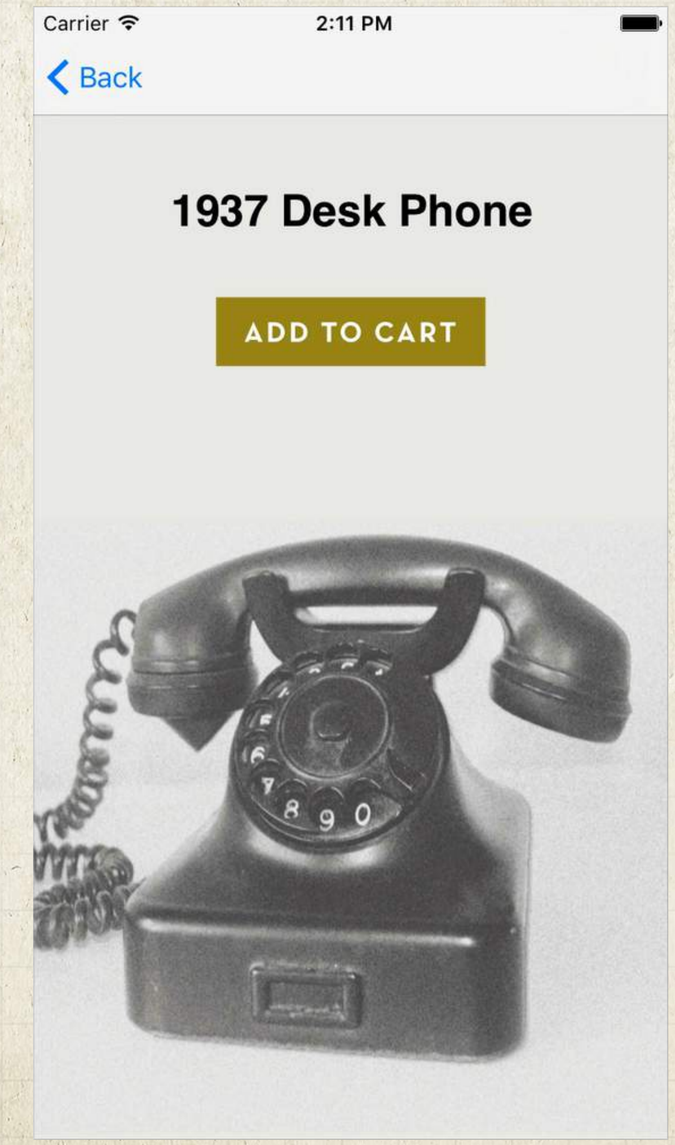
Swift is a new programming language introduced by Apple in June 2014 for creating iOS and OS X apps.

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

The App We're Going to Make





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

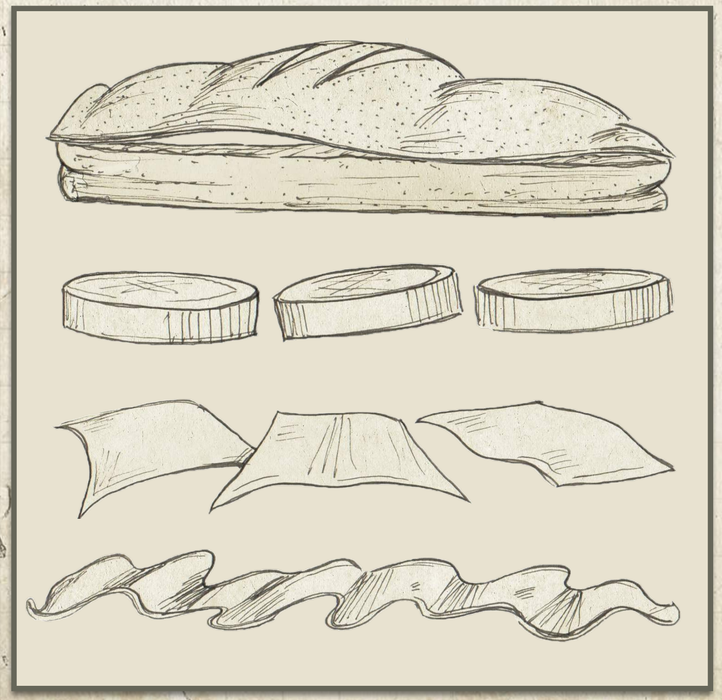
Section 1 - Creating an Xcode project

APP EVOLUTION with Swift

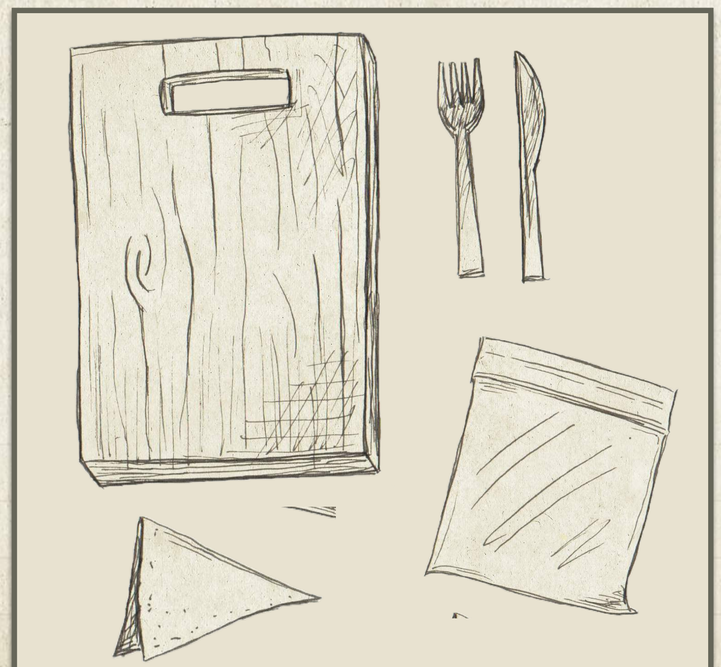
Apps and Sandwiches

what do we need to
make this sandwich?

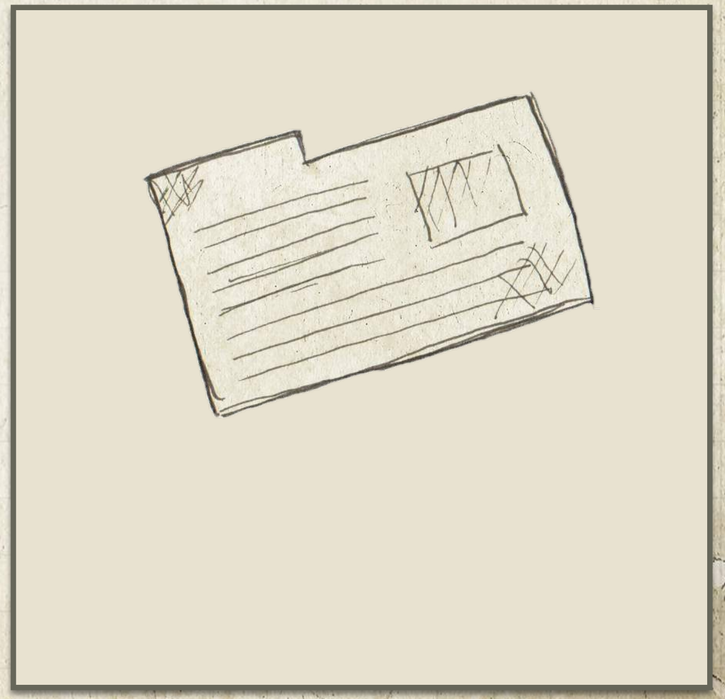
Raw Ingredients



Cooking Utensils



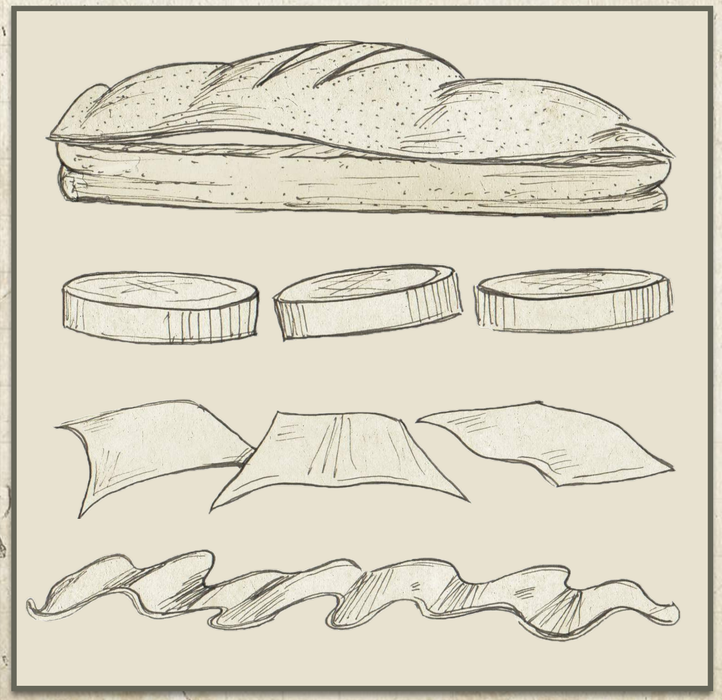
The Recipe



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

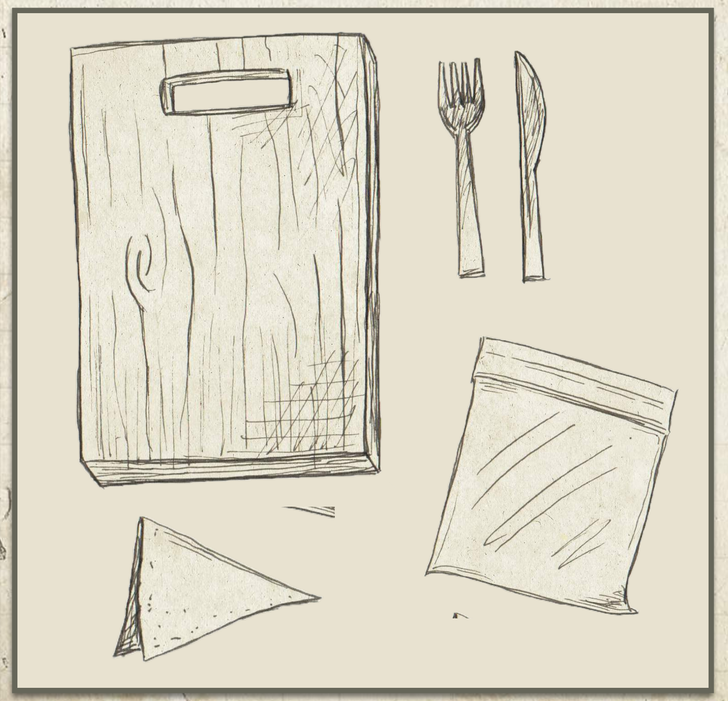
ADD TO CART

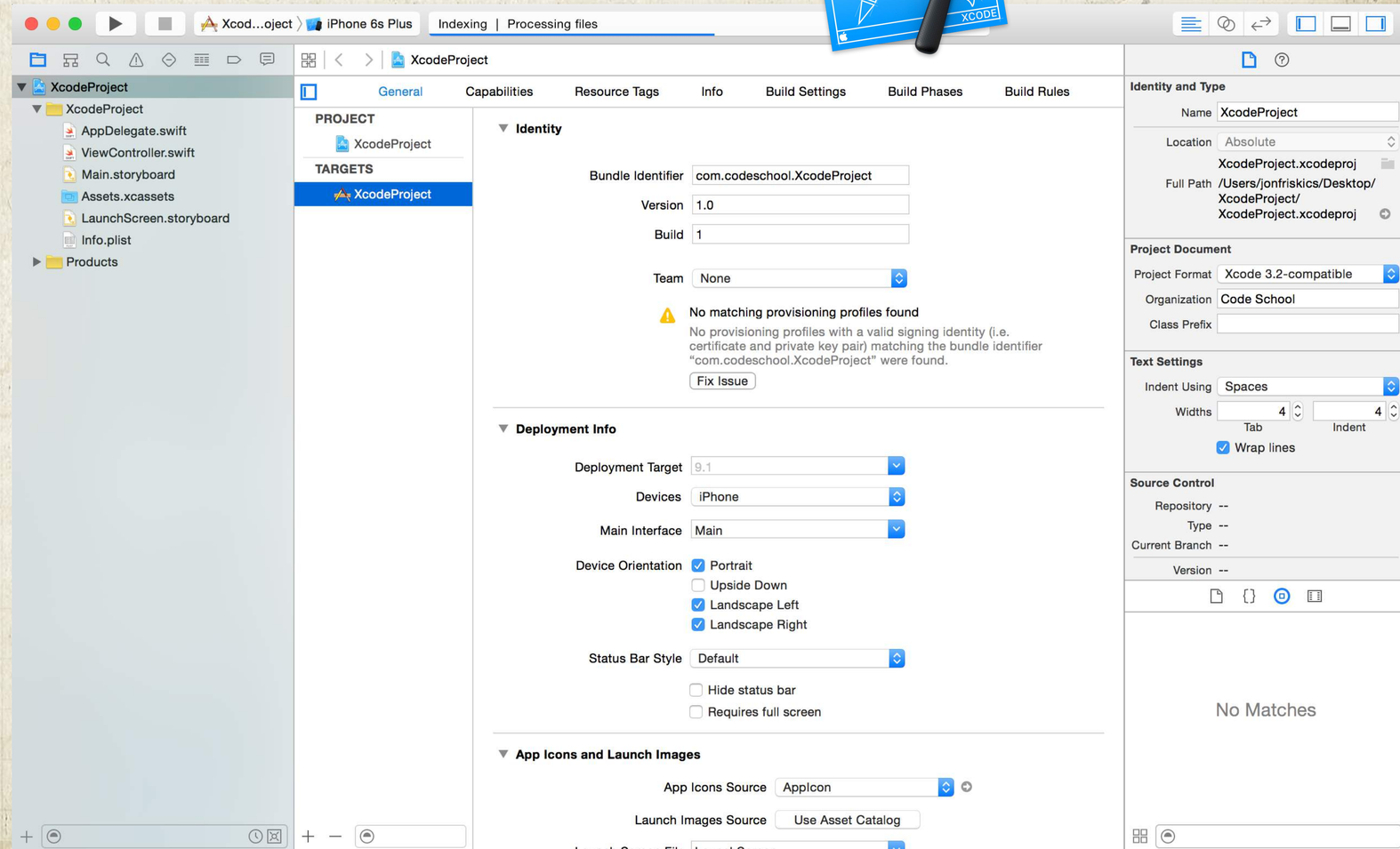
APP EVOLUTION with Swift

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

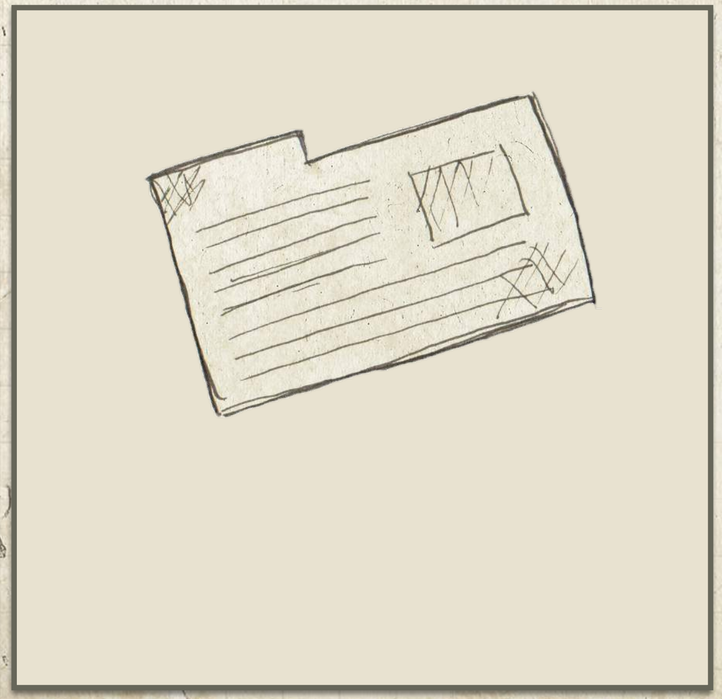




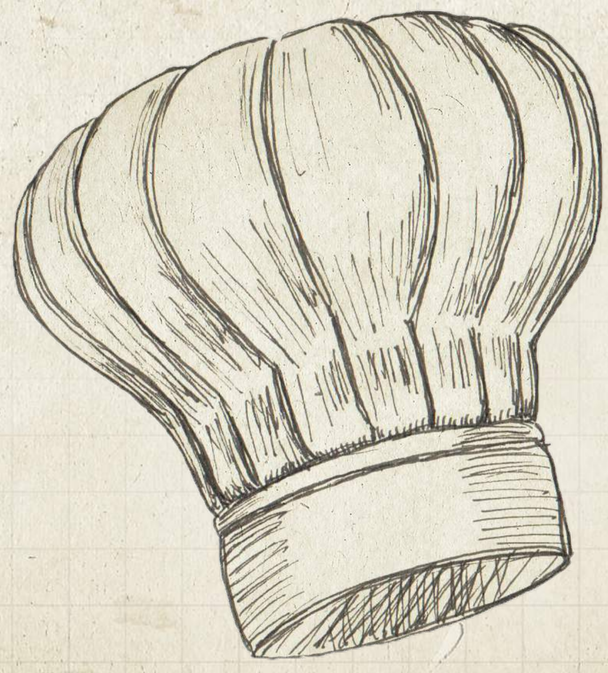
Swift Is Like the Instructions for Making the Sandwich

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!

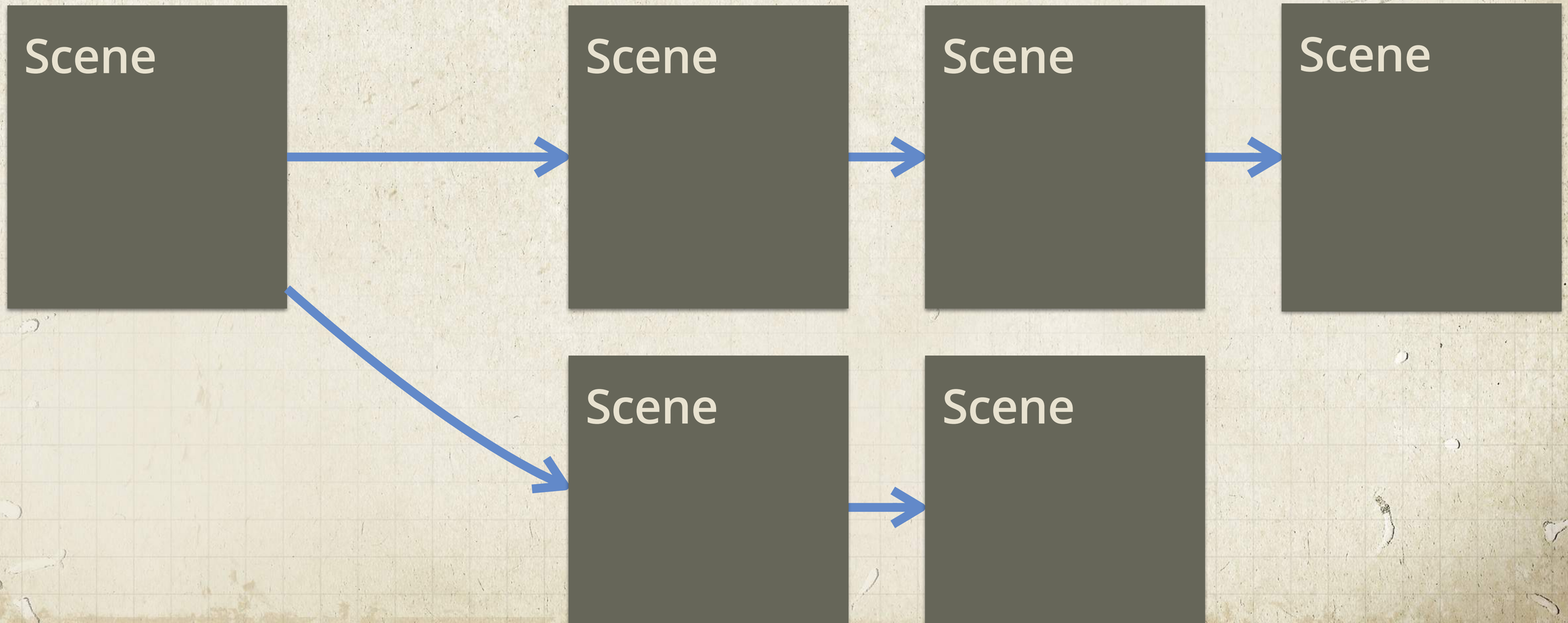


APP EVOLUTION with Swift

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

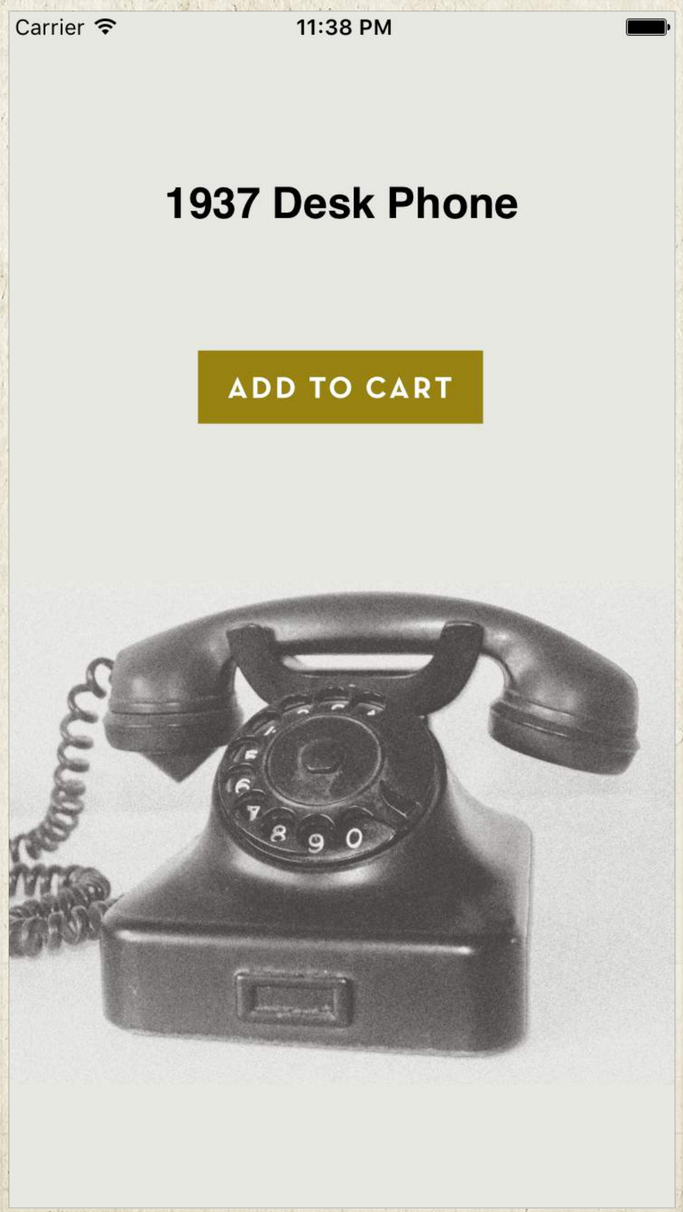
Level 1

Xcode and Storyboards

Section 2 - Storyboards and Subviews

APP EVOLUTION with Swift

Our First Goal: A Product Information Screen



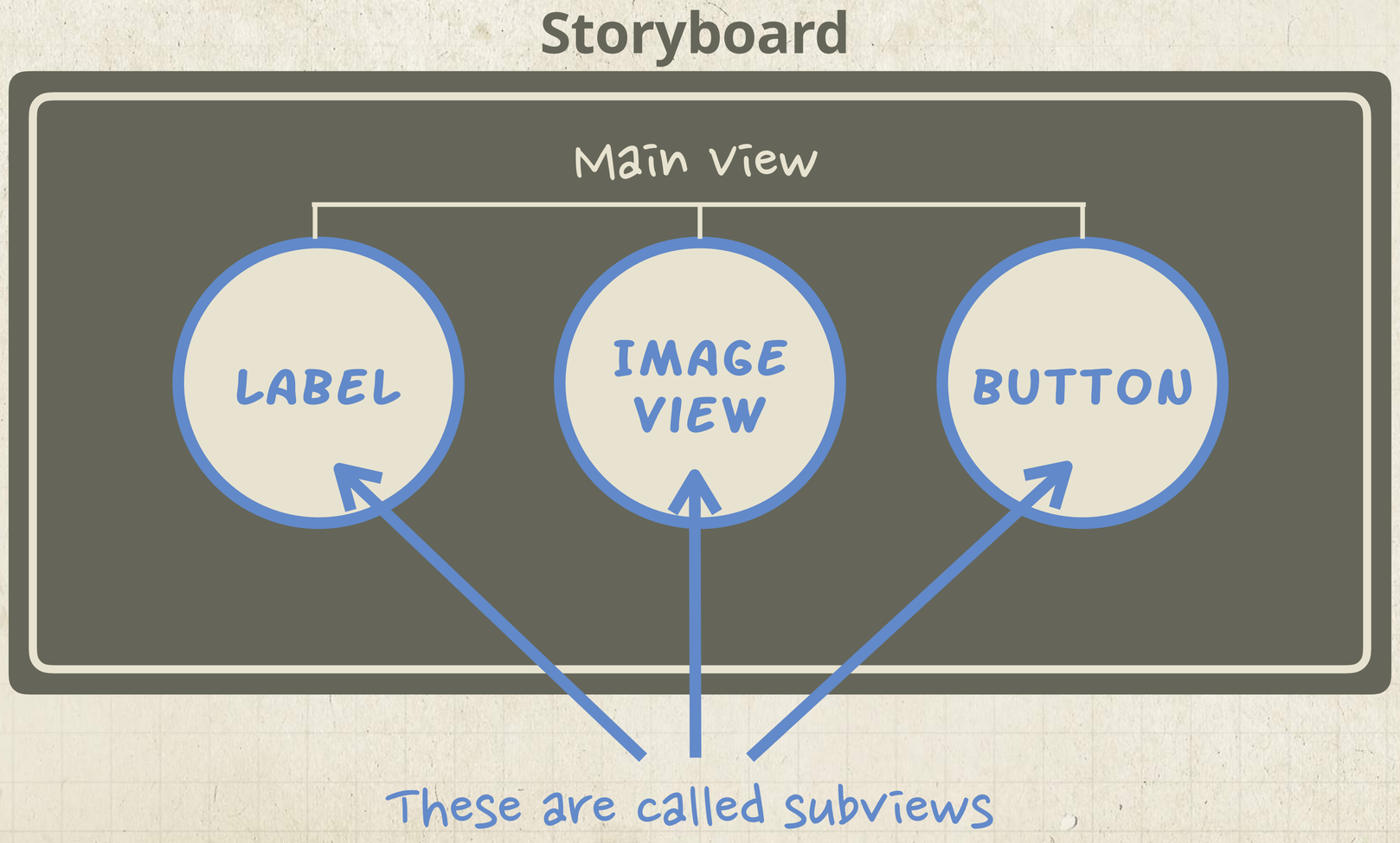
APP EVOLUTION with Swift

Screencast: Linking a Swift File to the Storyboard

APP EVOLUTION with Swift

Adding Subviews 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

APP EVOLUTION with Swift

Screencast: Displaying and Styling a Button

APP EVOLUTION with Swift

Level 2

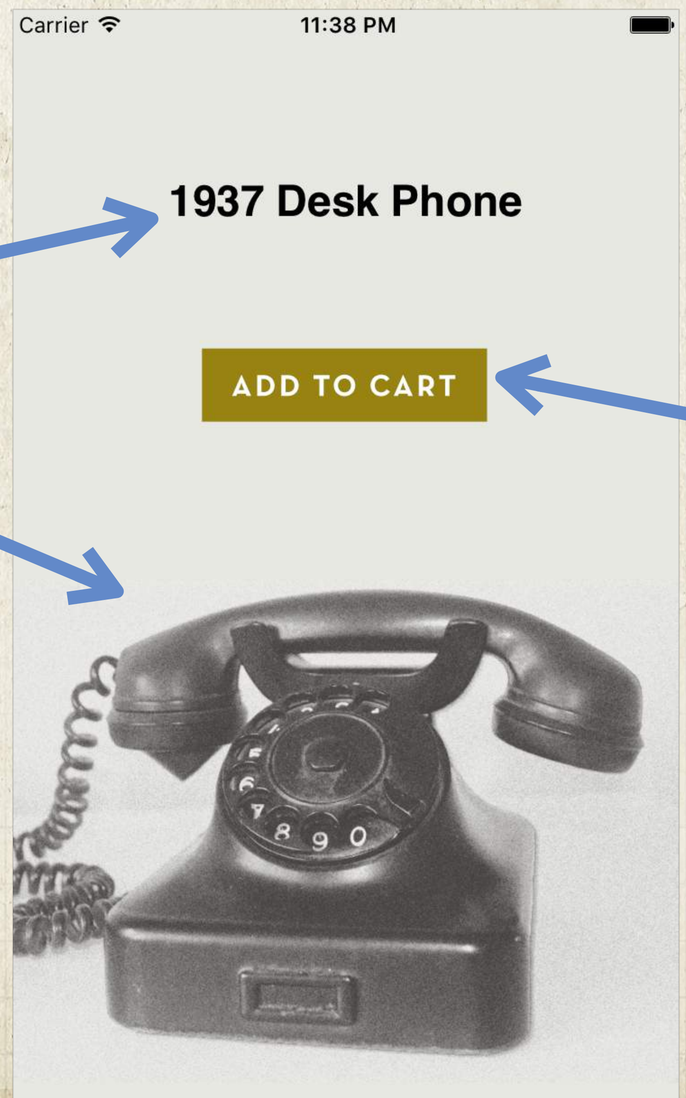
Outlets and Actions

Section 1 - Storyboard Outlets

APP EVOLUTION with Swift

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

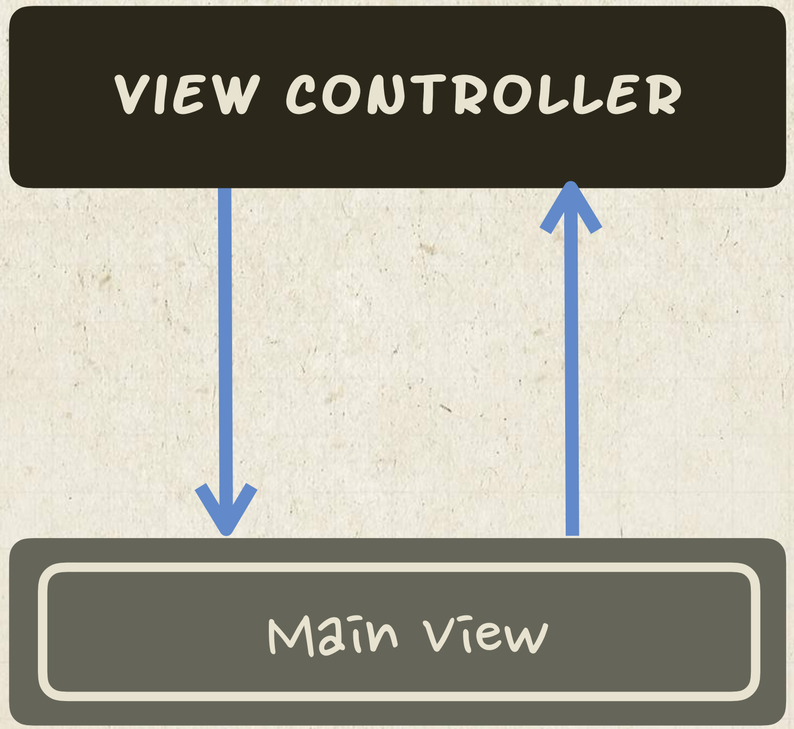
Make this button do something when it is tapped

APP EVOLUTION with Swift

Why View Controllers?

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

controller can tell view what to display

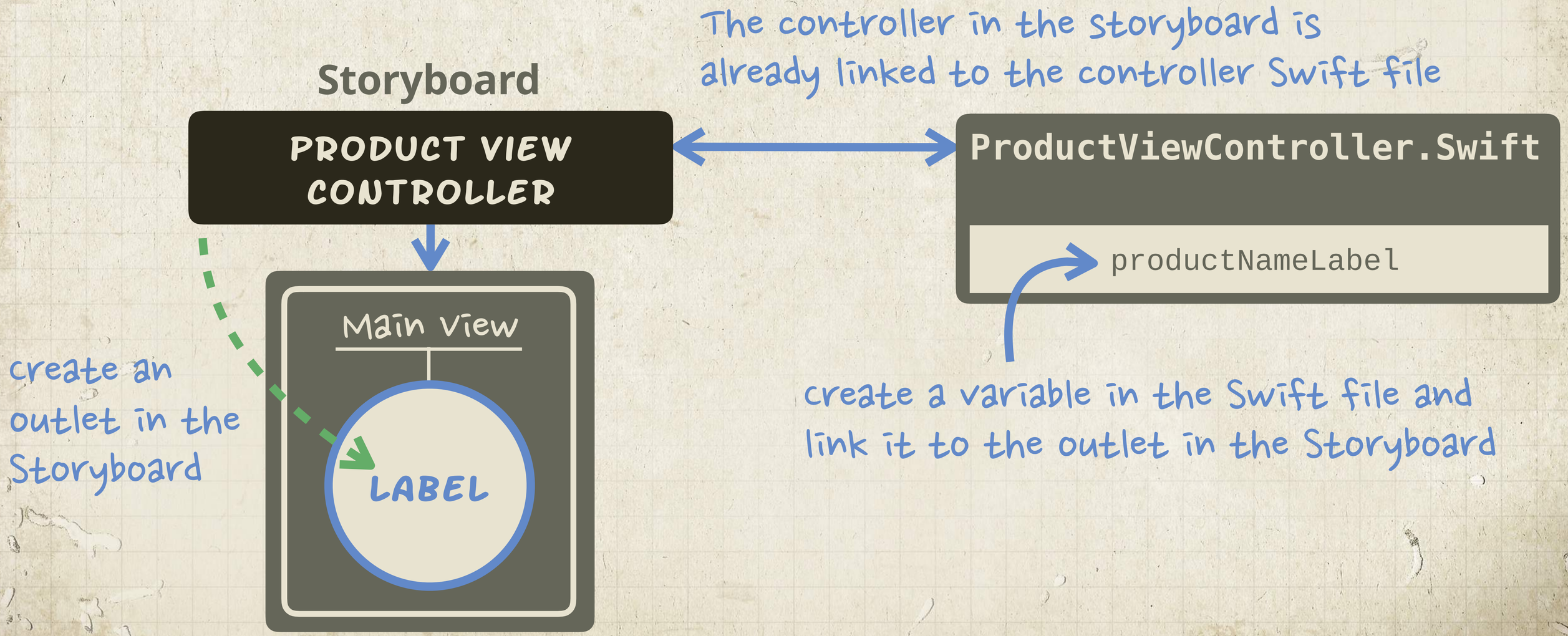


view can tell controller when user is interacting

Screencast: Creating a Swift File

How Outlets Work

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



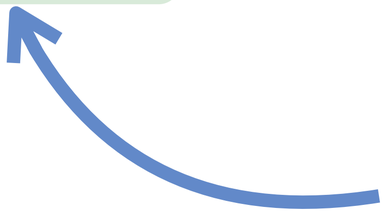
Screencast: Using the Assistant Editor

APP EVOLUTION with Swift

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

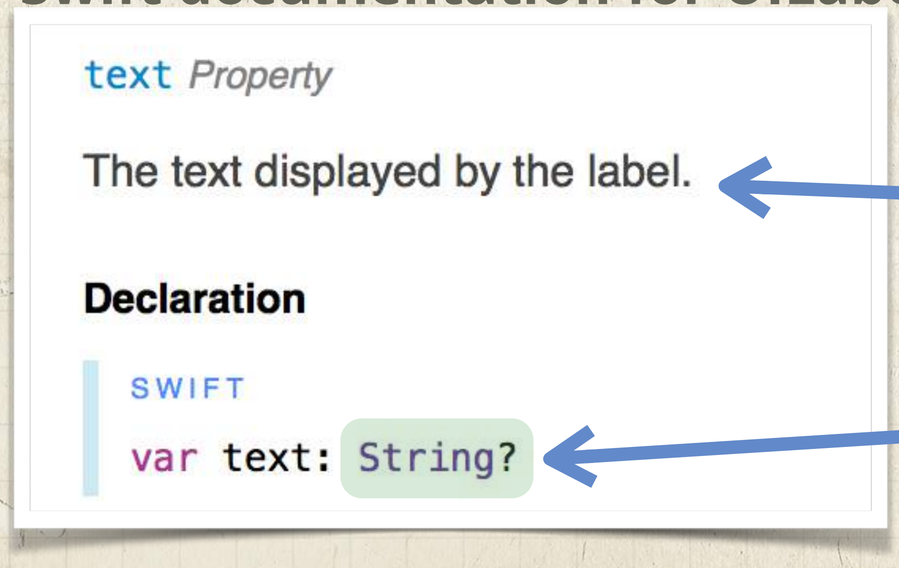
Finding Properties in Apple's Documentation

ProductViewController.swift

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

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

Swift documentation for UILabel



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")
    }

}
```

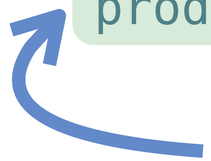


Image views have an image property that you can set equal to a **UIImage** object

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

SWIFT

```
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

Demo: Image View Being Set With Code

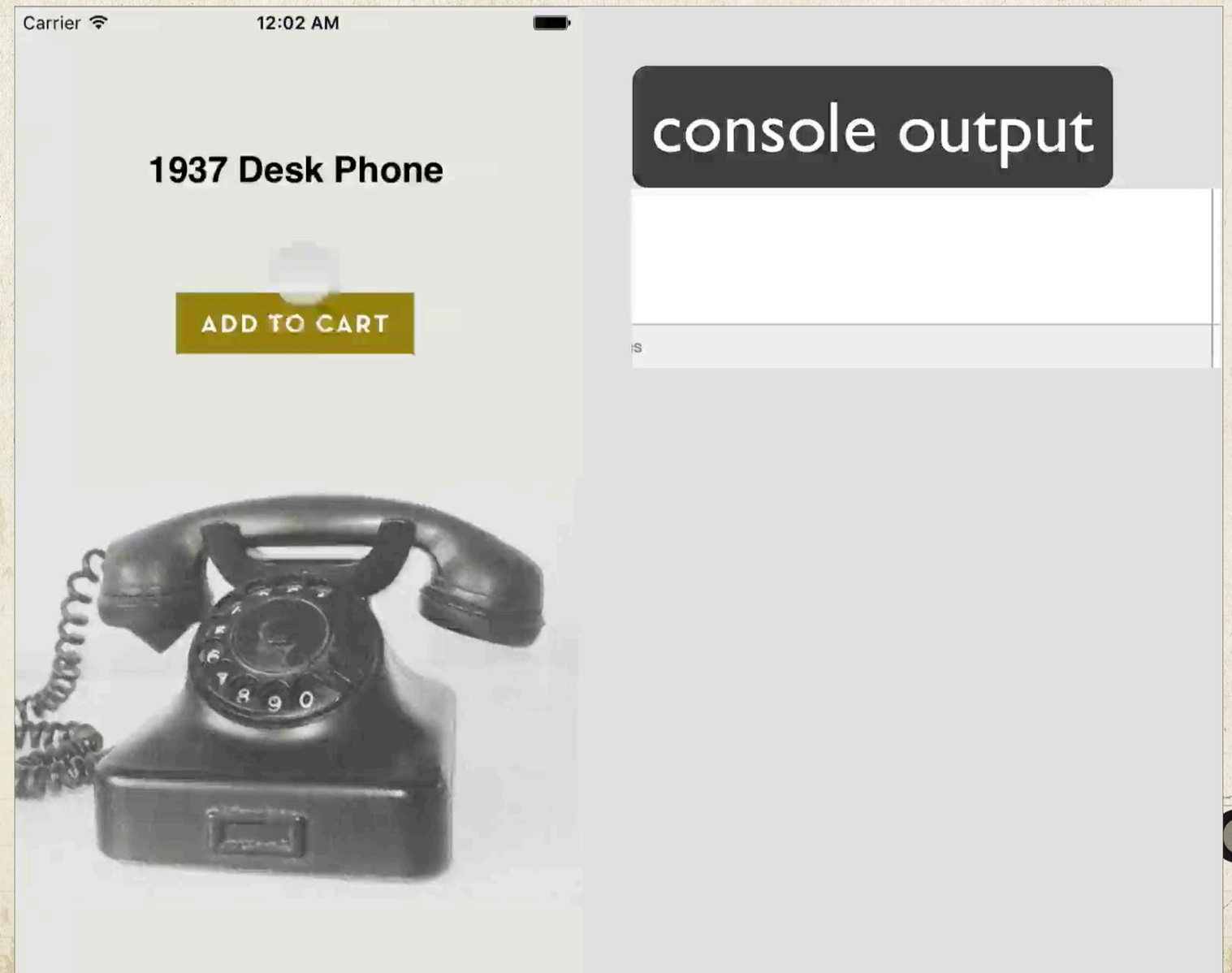
APP EVOLUTION with Swift

Level 2

Outlets and Actions

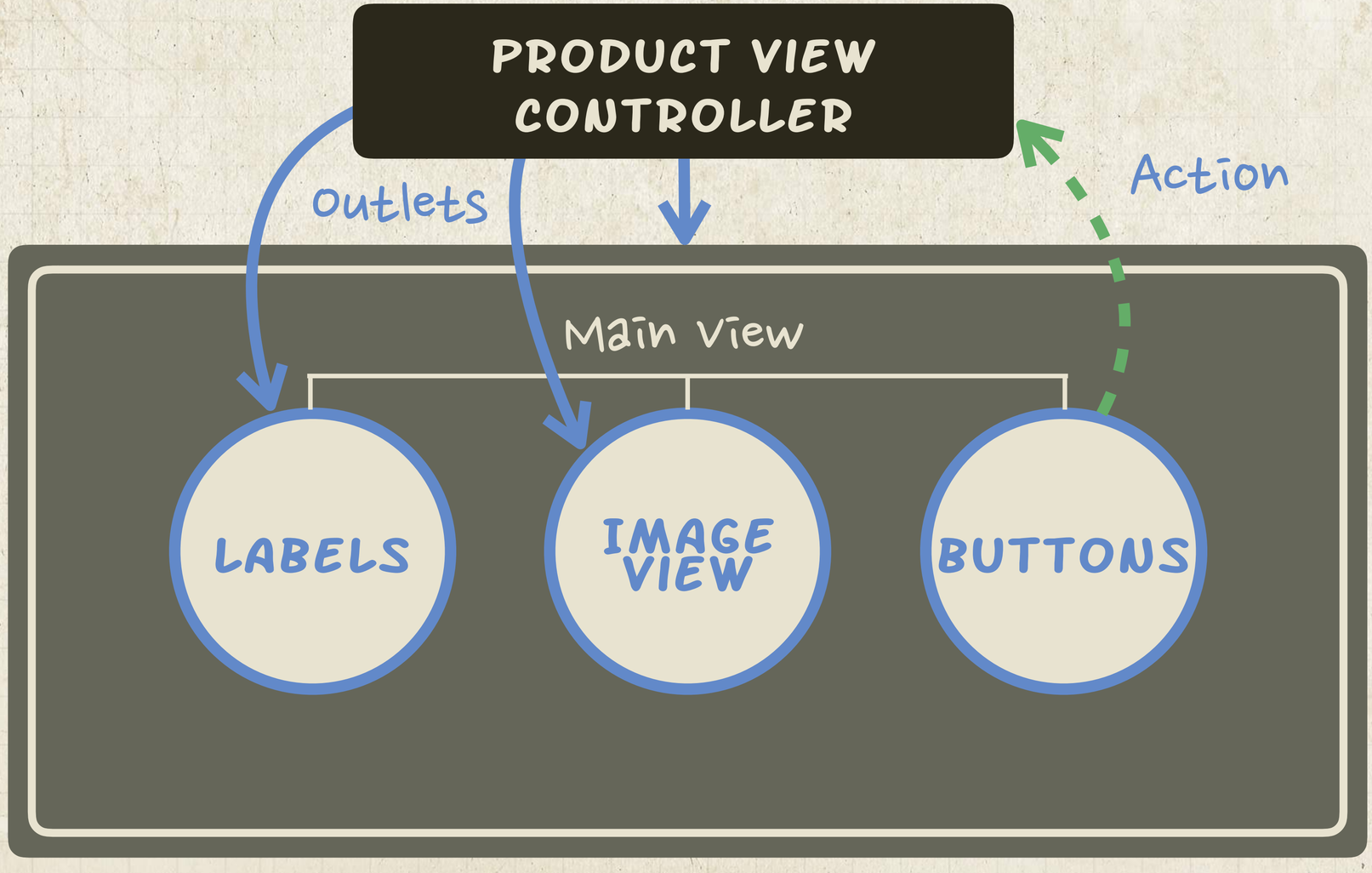
Section 2 - Storyboard Actions

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 {

    }

}
```

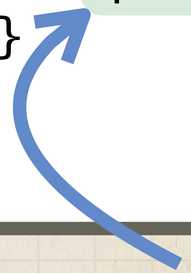
Void means the function doesn't return any values

This function now runs when the button is tapped

Logging a Message When the Button Is Tapped

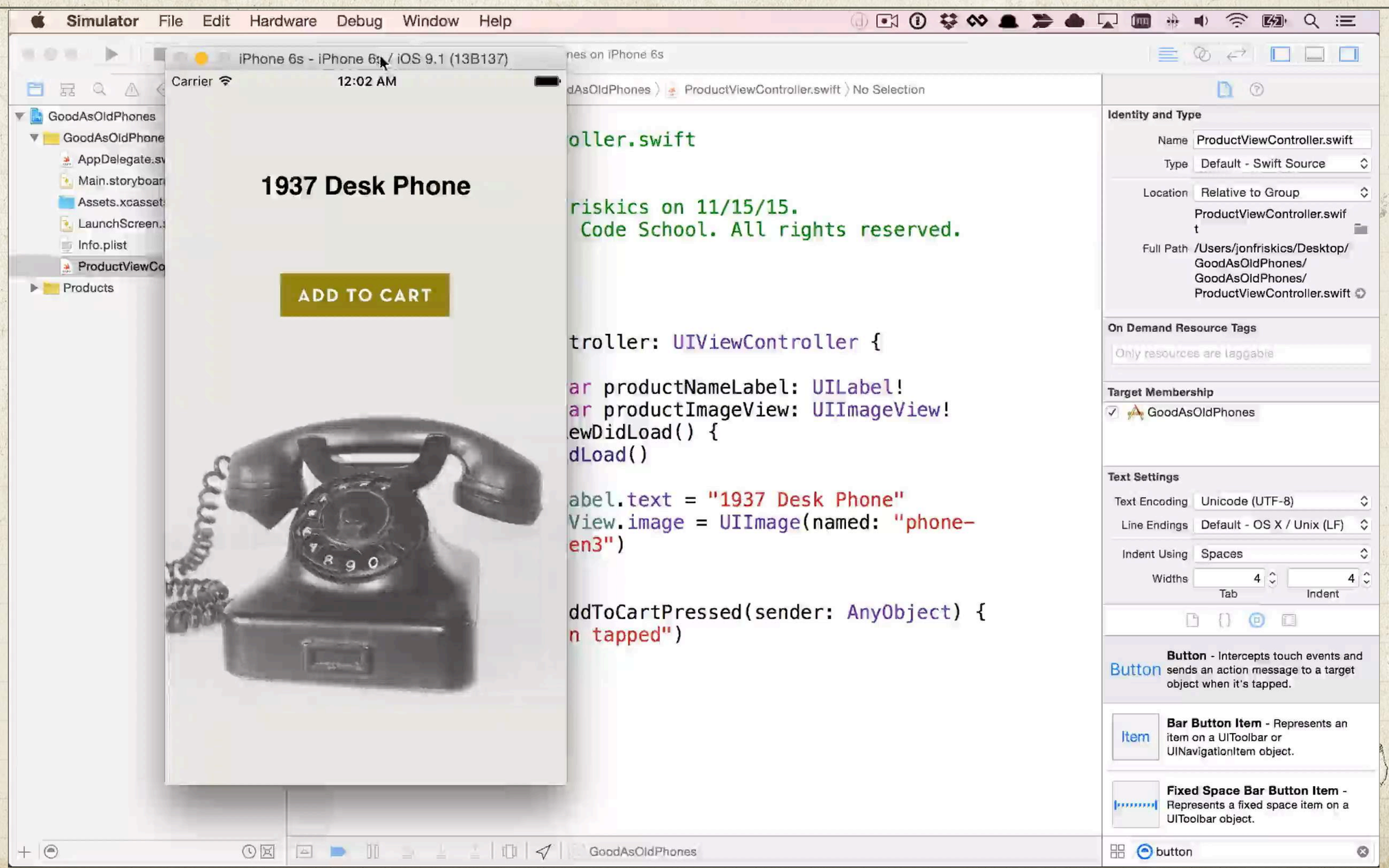
ProductViewController.swift

```
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

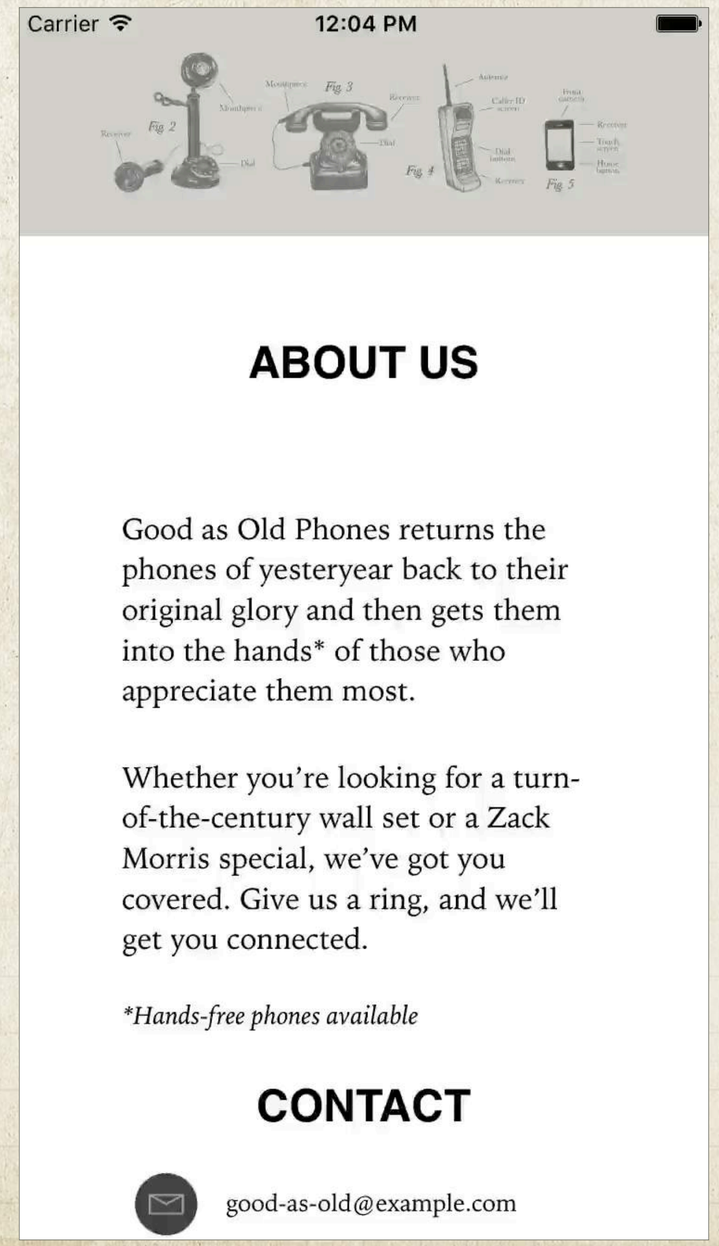
Demo: Button Tap Logs a Message to the Console



Level 3

Scroll Views

Demo: Content Scrolling on the Screen

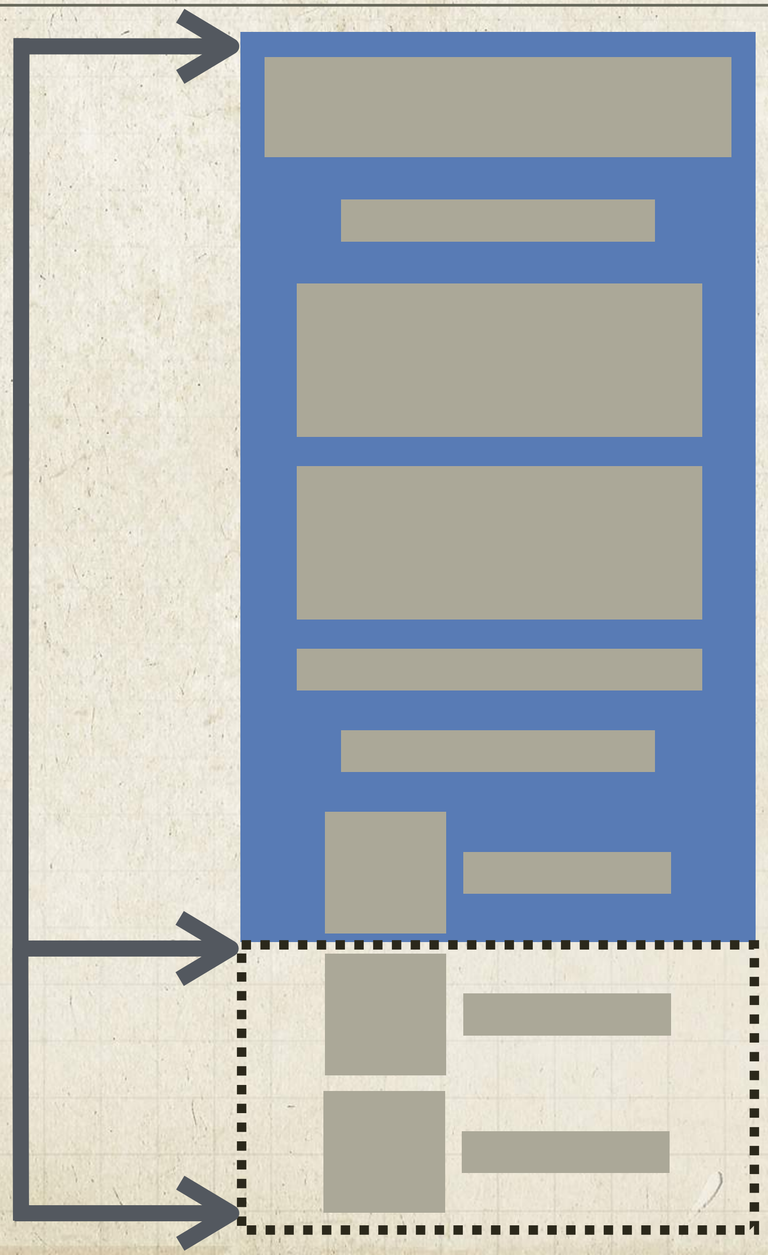


APP EVOLUTION with Swift

How a Scroll View Works

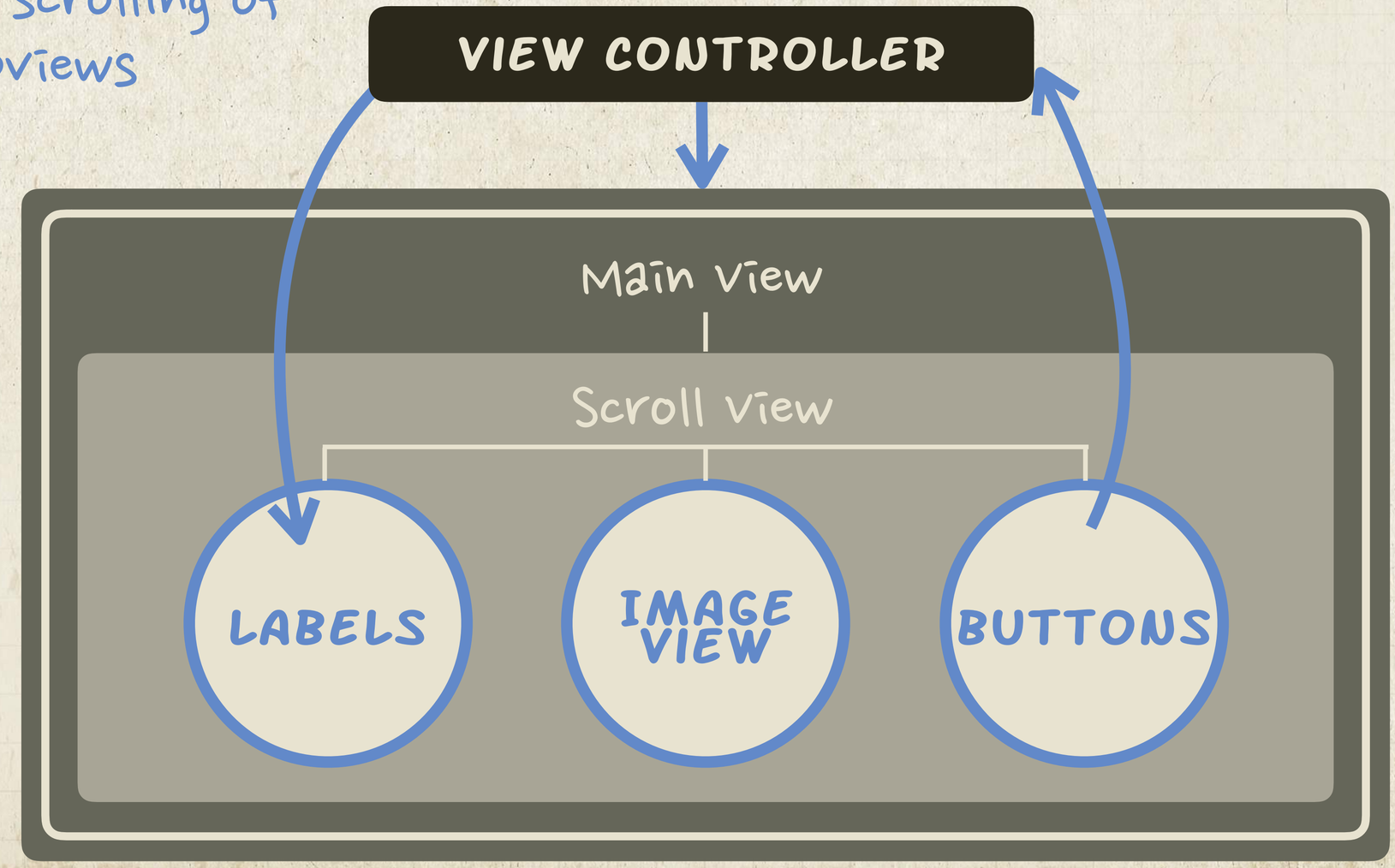
The available space on the iPhone screen

Sometimes your content needs more space than that



Scroll Views Contain Any Subviews That Can Be Scrolled

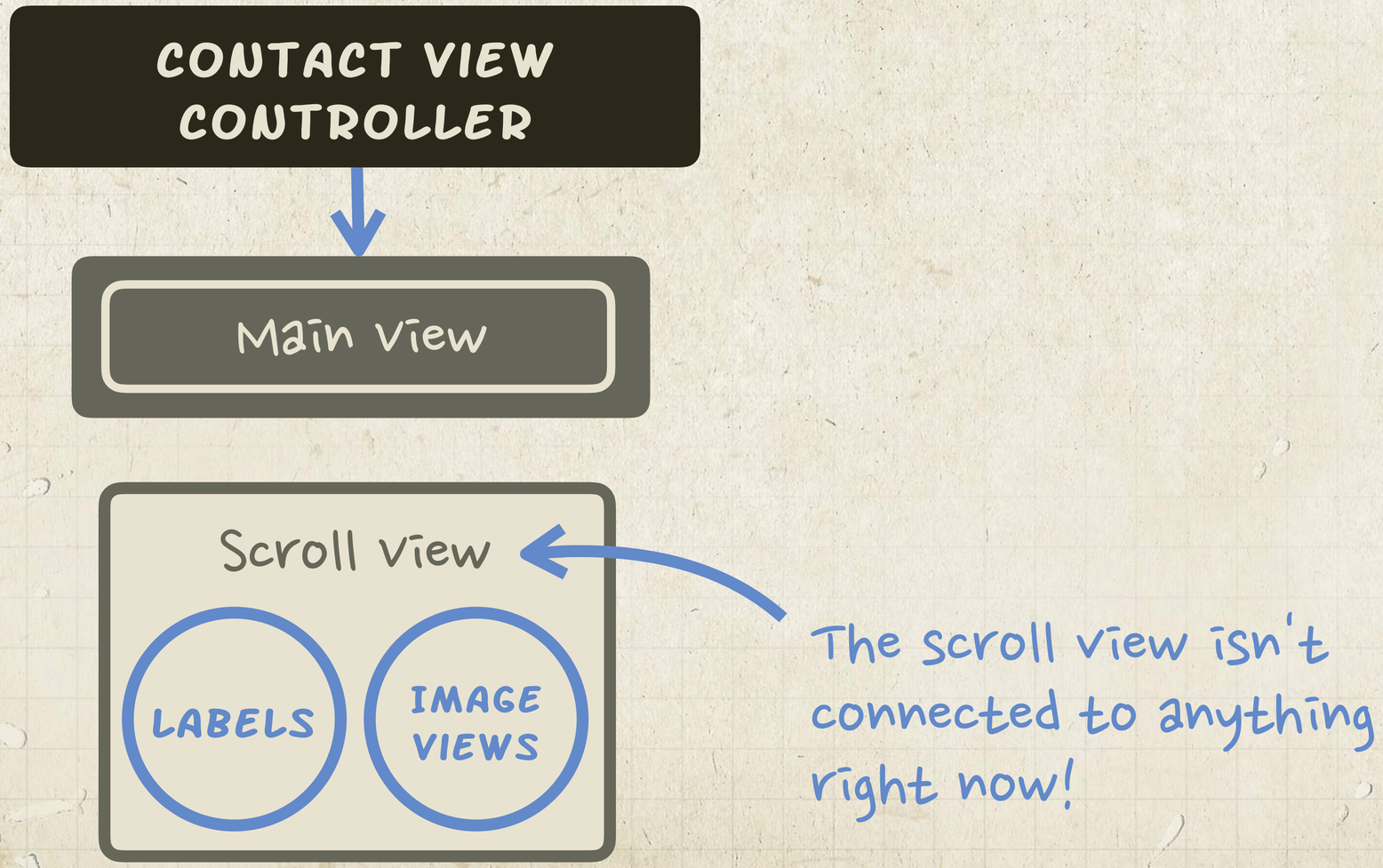
A **UIScrollView** object manages the scrolling of all of its subviews



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.

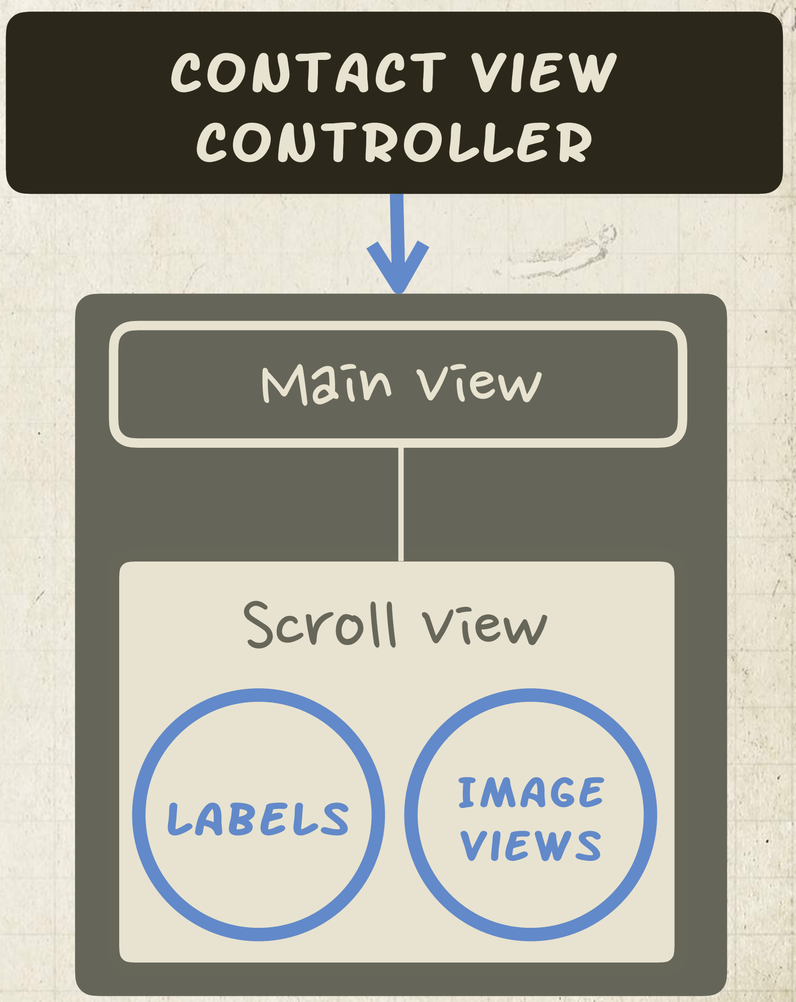


Connecting the Scroll View to the Main View

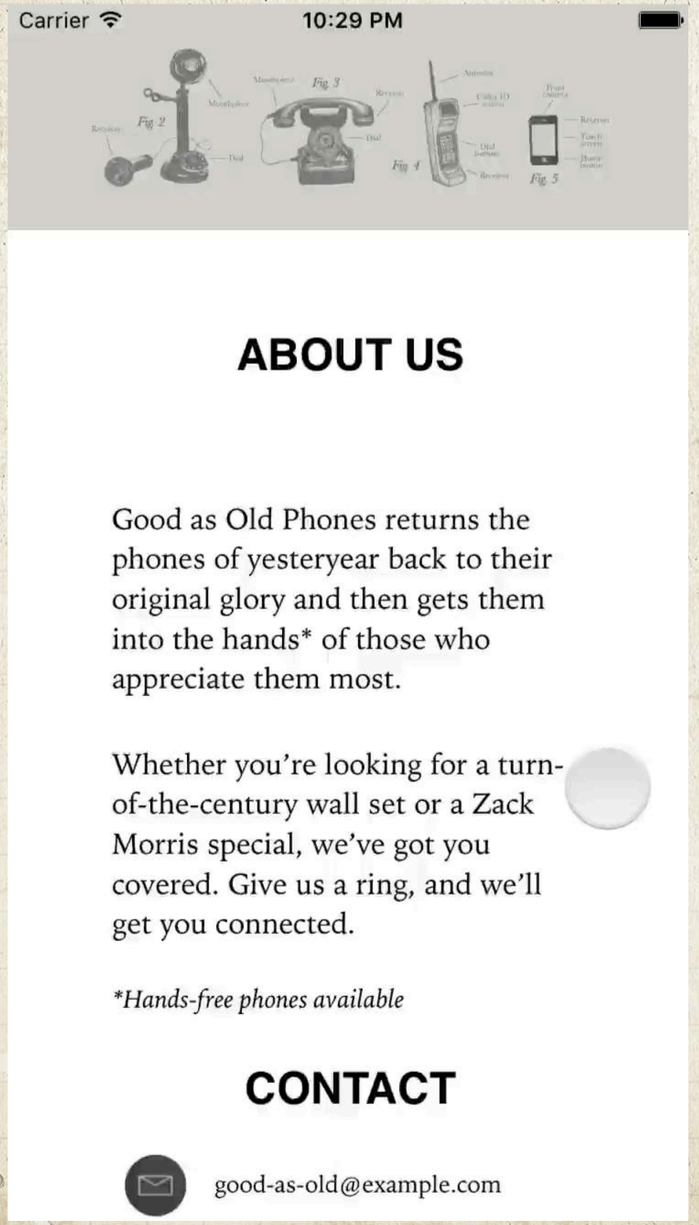
ContactViewController.swift

```
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



Problem: The Scroll View Isn't Scrolling



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

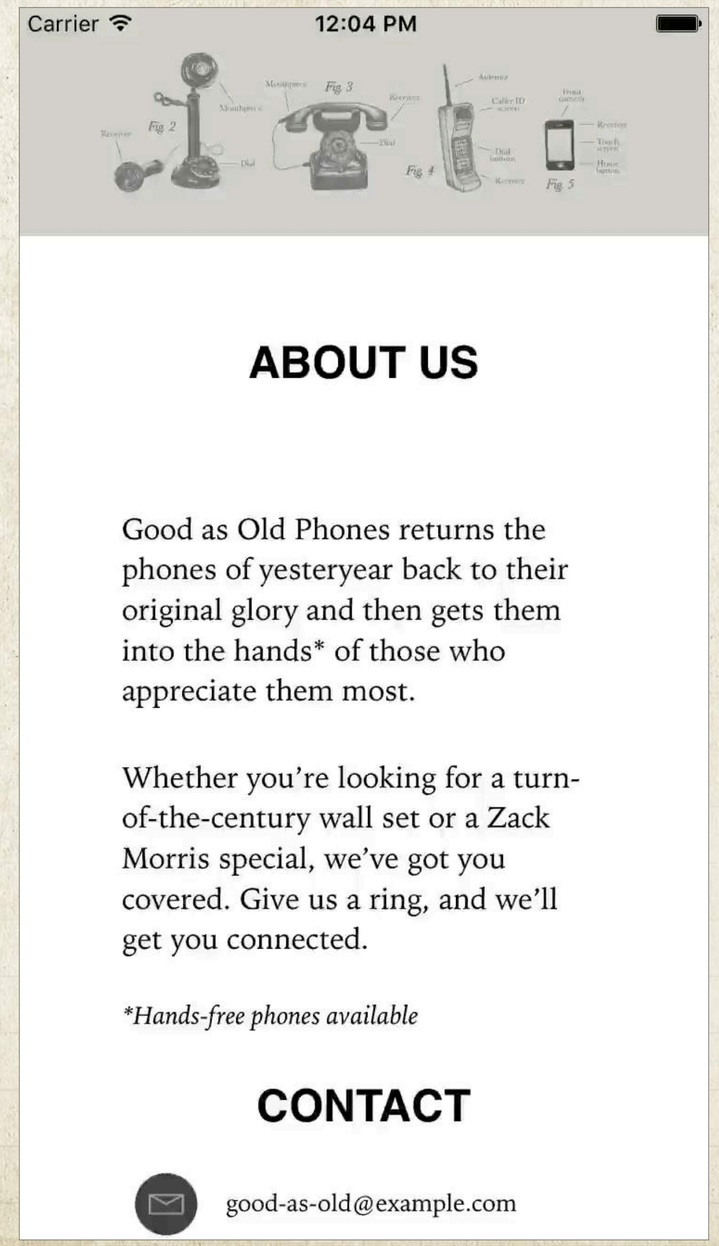
Setting the Scroll View's Content Size

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 = CGSizeMake(375, 800)
    }
}
```

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

Demo: The Working Scroll View



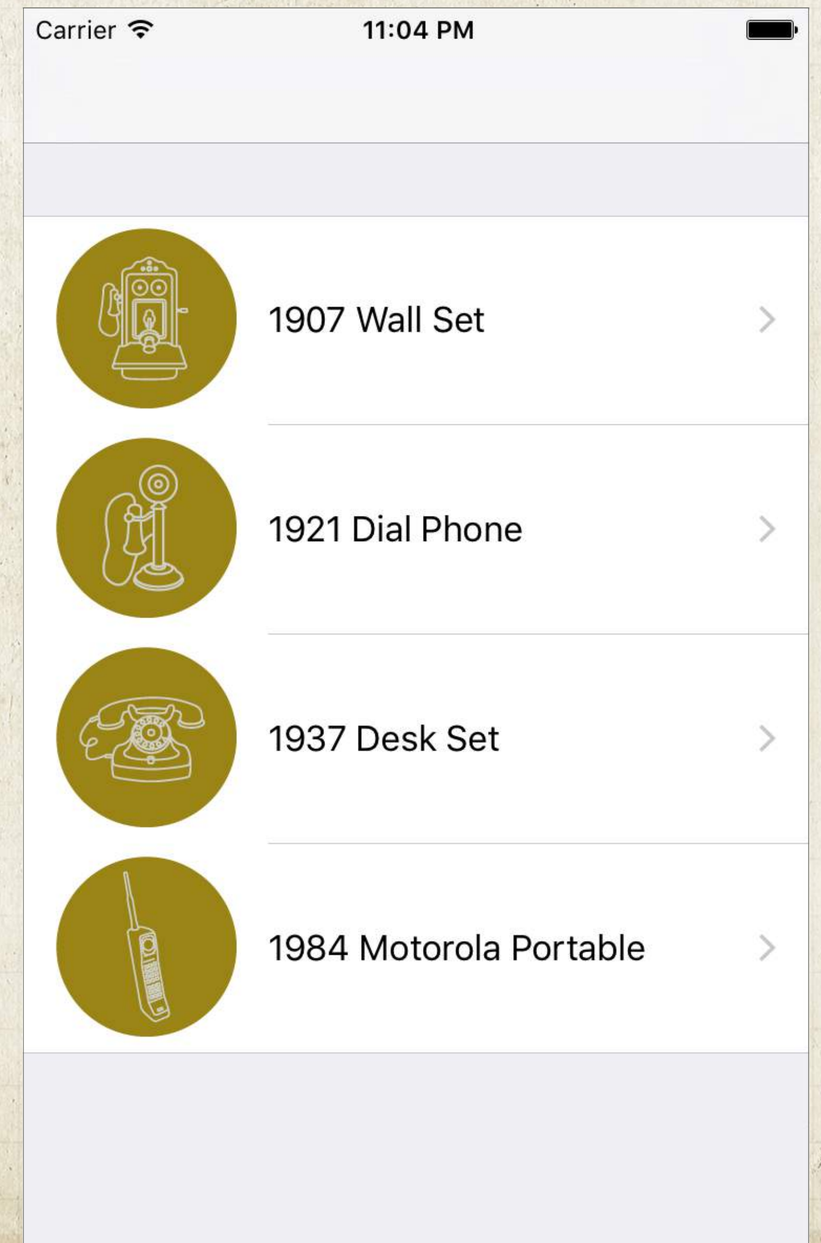
APP EVOLUTION with Swift

Level 4

Table Views

Section 1 - Creating a Table View

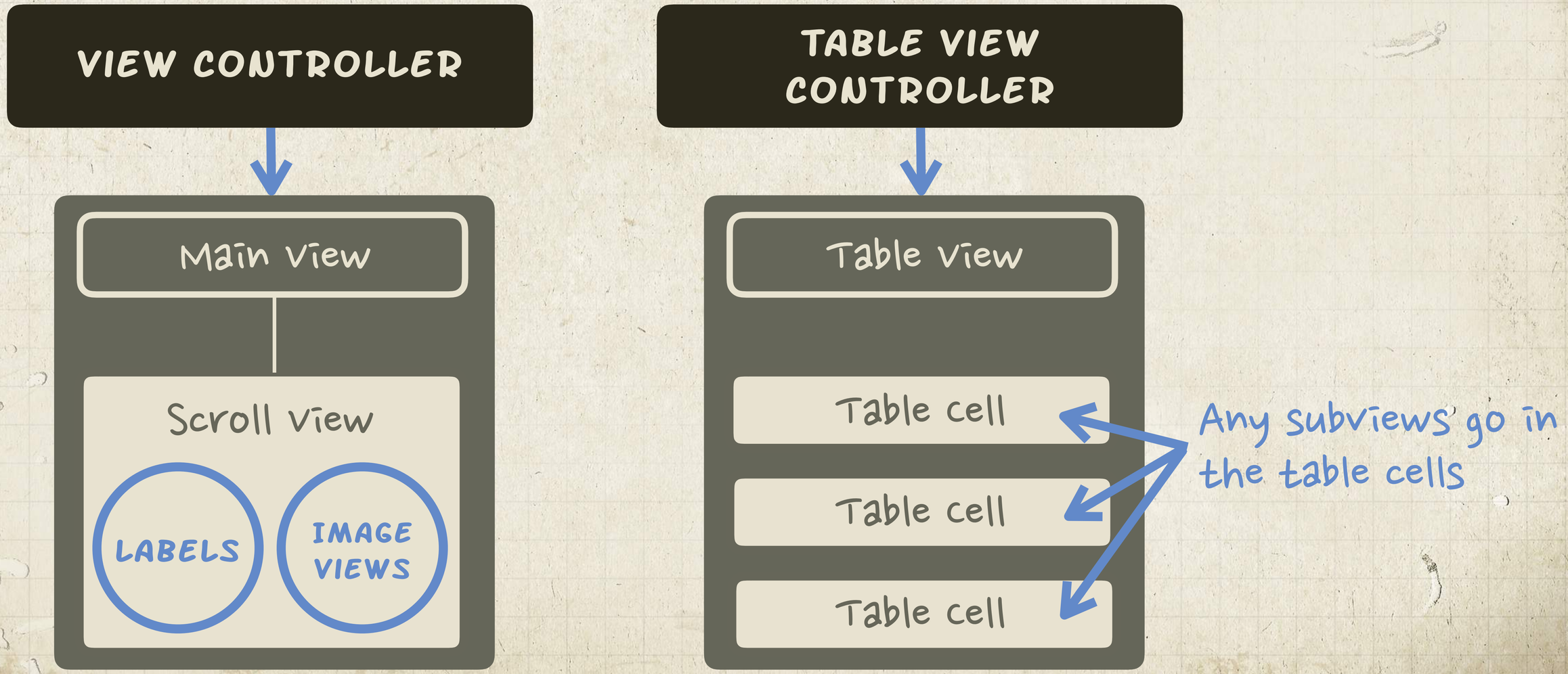
Demo: Using a Table View to Display a List of Things



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



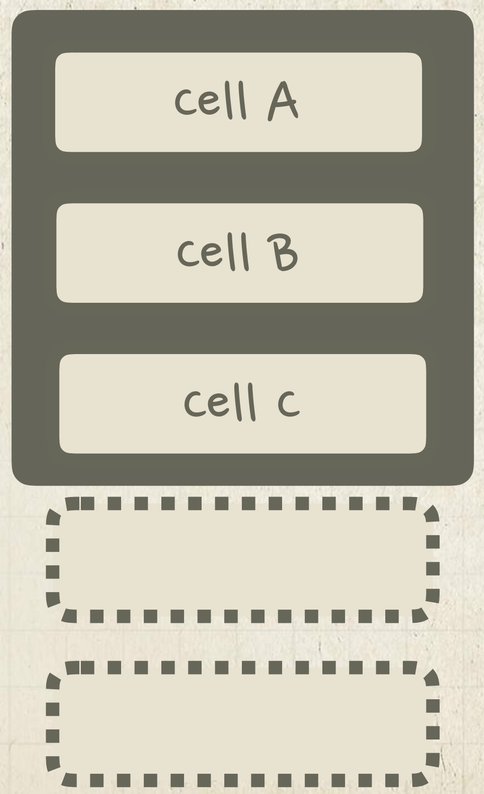
Screencast: Creating a Table View Controller

APP EVOLUTION with Swift

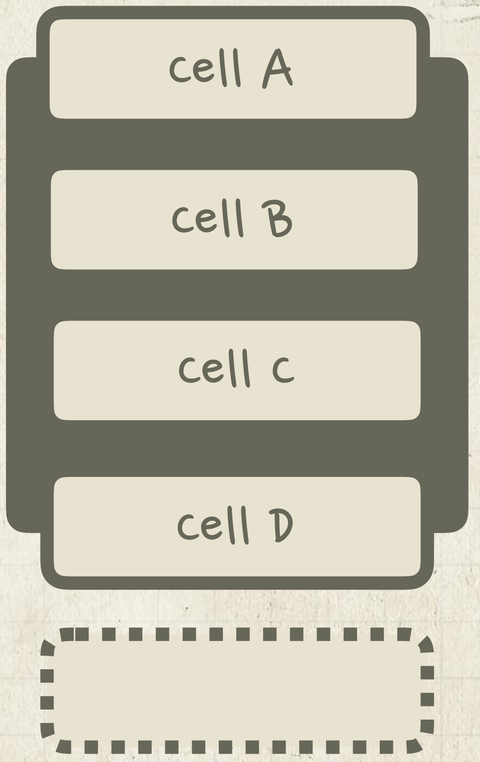
How Table View Cell Recycling Works

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



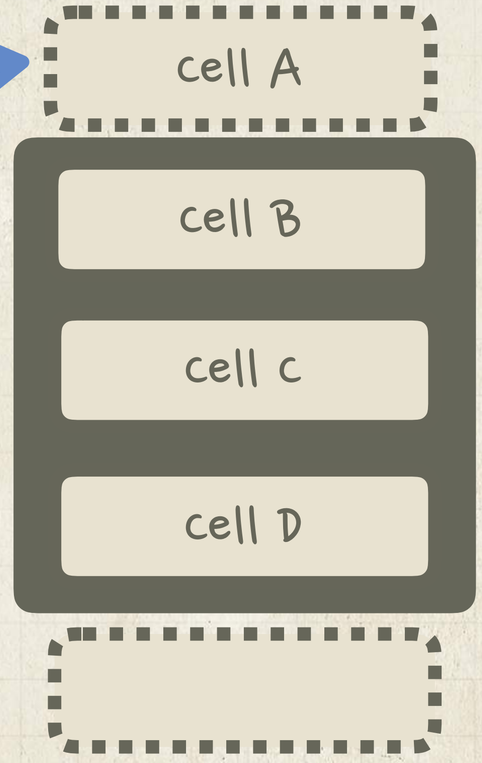
As the table view scrolls, now four unique cells are visible



How Table View Cell Recycling Works

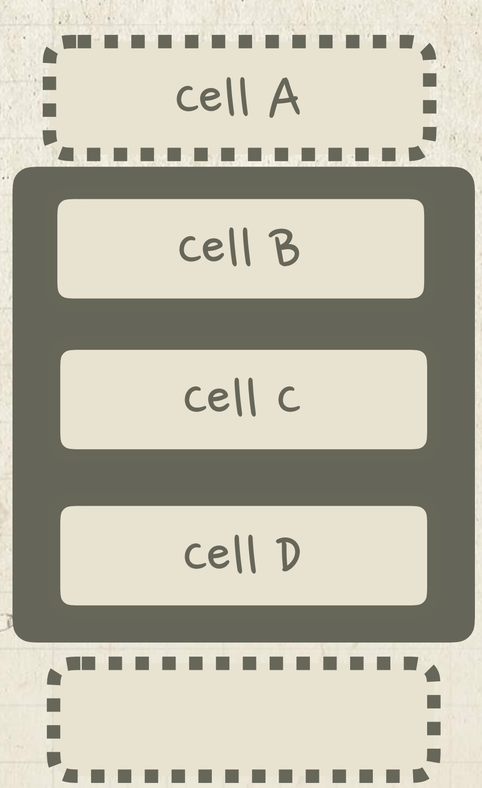
Table cells are recycled as they scroll offscreen, and any new cells are just recycled cells filled in with new data.

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

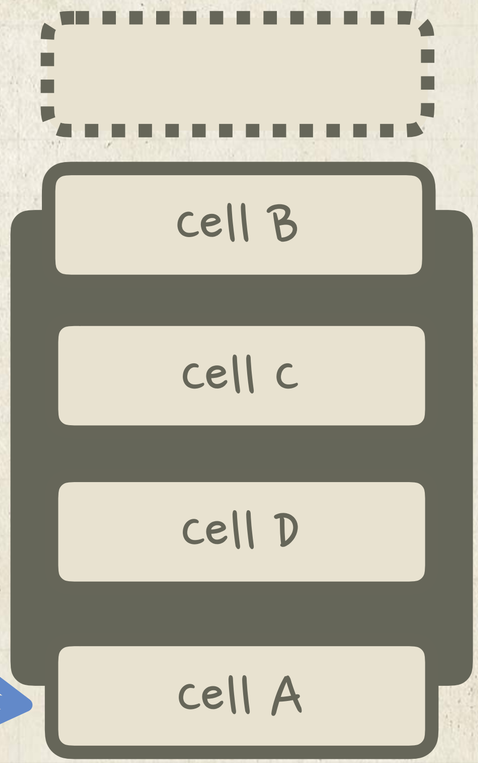


How Table View Cell Recycling Works

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

ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: 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
```

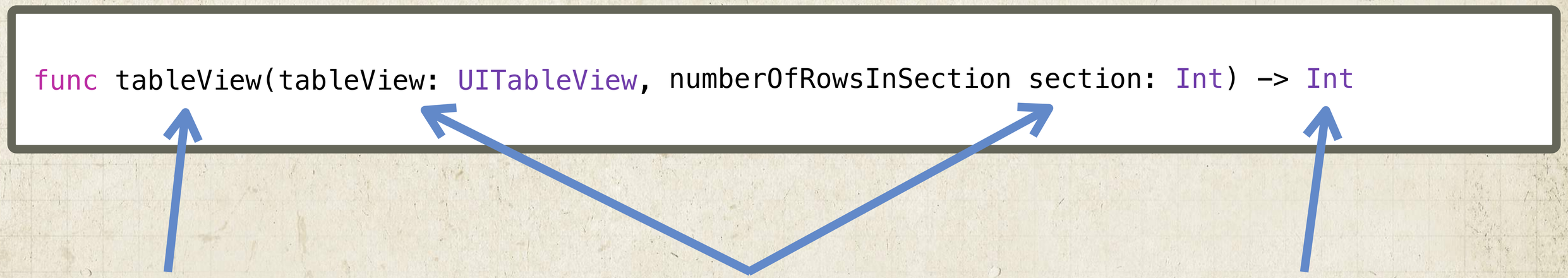
Set the number of rows

create one cell for each row

```
func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
```

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, cellForRowAtIndexPath indexPath: NSIndexPath) ->
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 function should return a number

This table view will have this many rows

Adding the Function That Will Create Cells

The cellForRowAtIndexPath 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
  { ... }

  override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {

  }
}
```

Runs one time for each row (5 times in our case)

This function should return a cell

Creating a Cell for Each Row

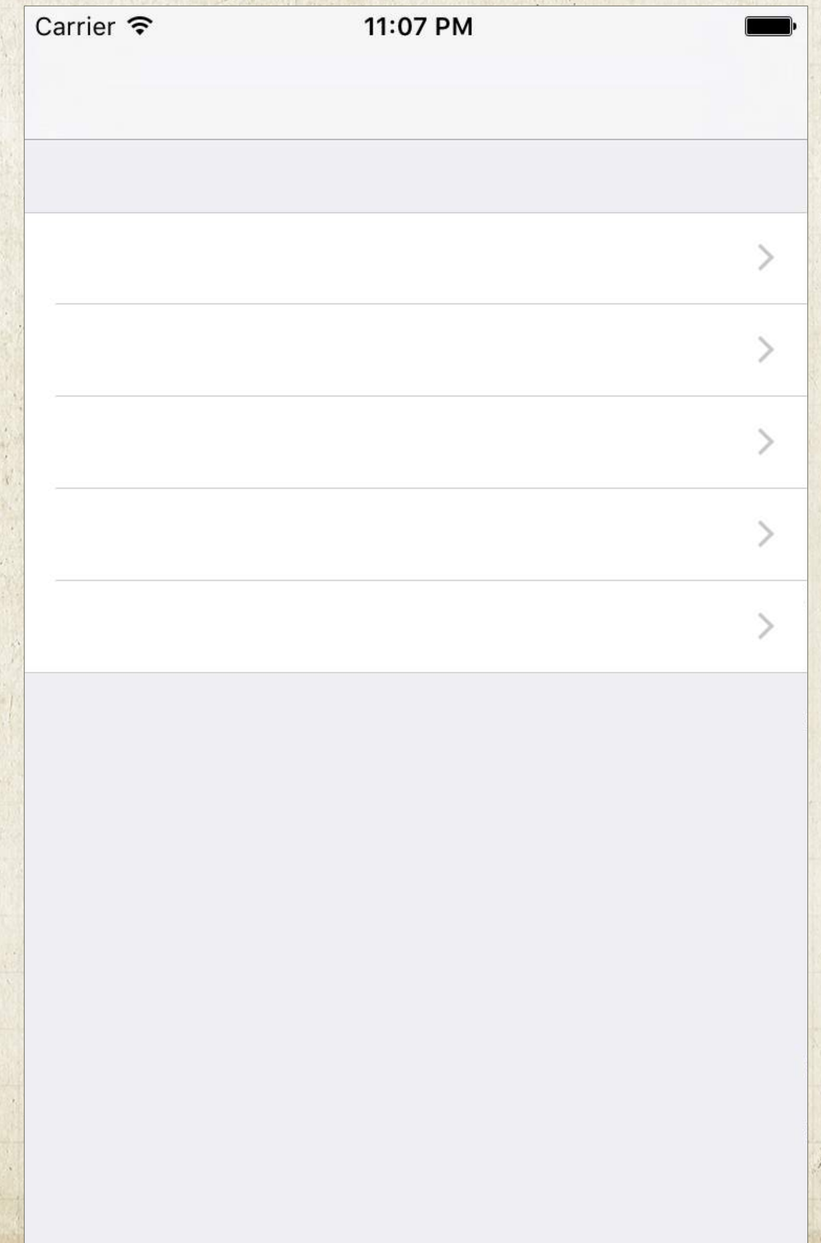
ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {

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

  override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("ProductCell",
                                                           forIndexPath: indexPath)

    return cell
  }
}
```

Return the created cell

This matches the identifier we added in the Storyboard

Demo: Table View With Cells



APP EVOLUTION with Swift

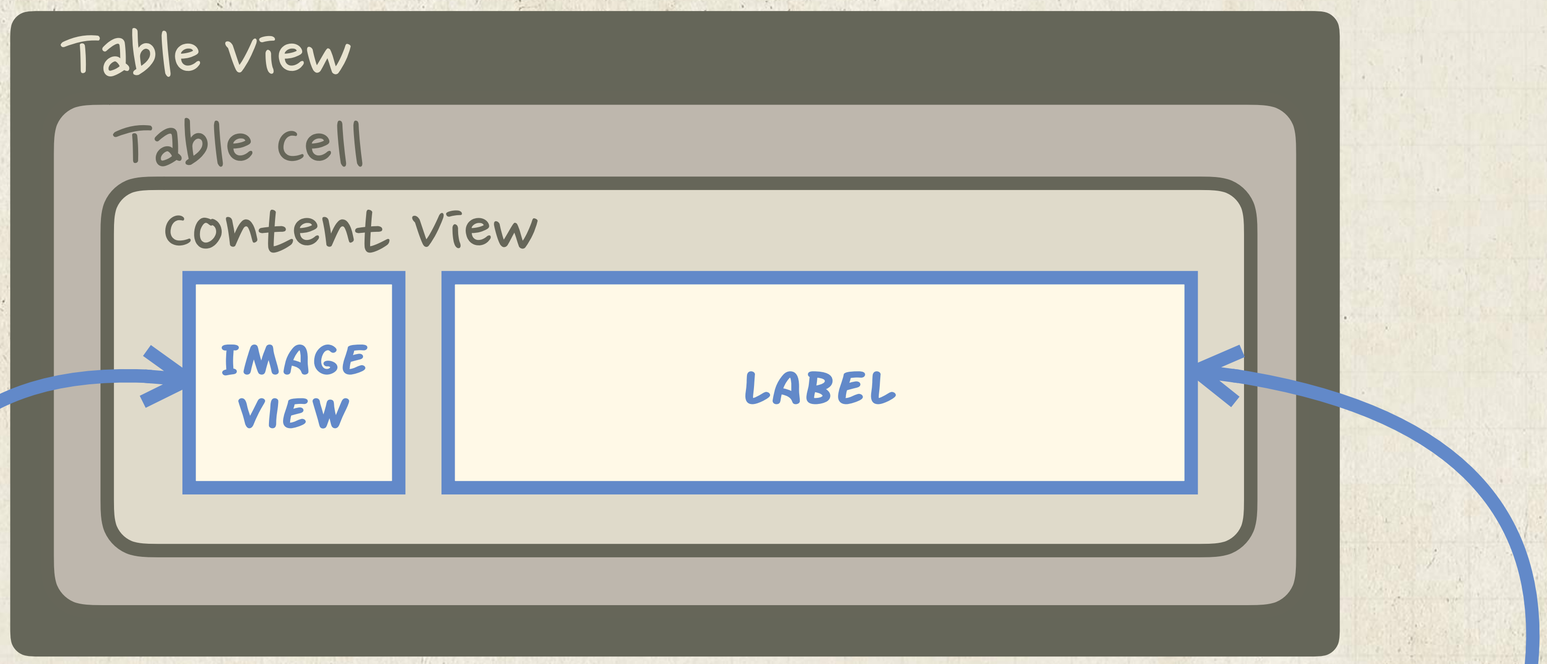
Level 4

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,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("ProductCell",
                                                           forIndexPath: 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

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

This means it's **optional**.

It may or may not exist

APP EVOLUTION with Swift

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"
```

So setting it will crash the app!

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

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

APP EVOLUTION with Swift

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"

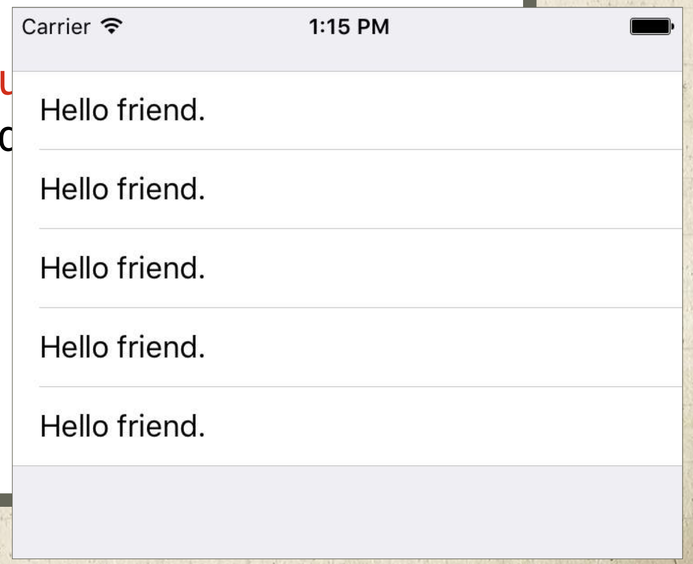
textLabel exists → continue running the line of code

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,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("Produ
                                                          forInd

    cell.textLabel?.text = "Hello friend"

    return cell
  }
}
```

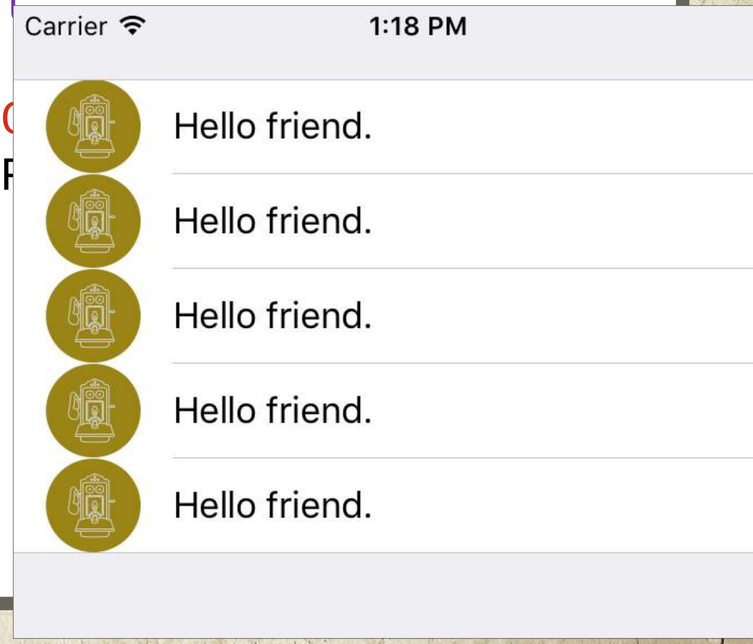


Setting the imageView of a Cell

ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
  ...

  override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("ProductC
                                                          forIndexP

    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

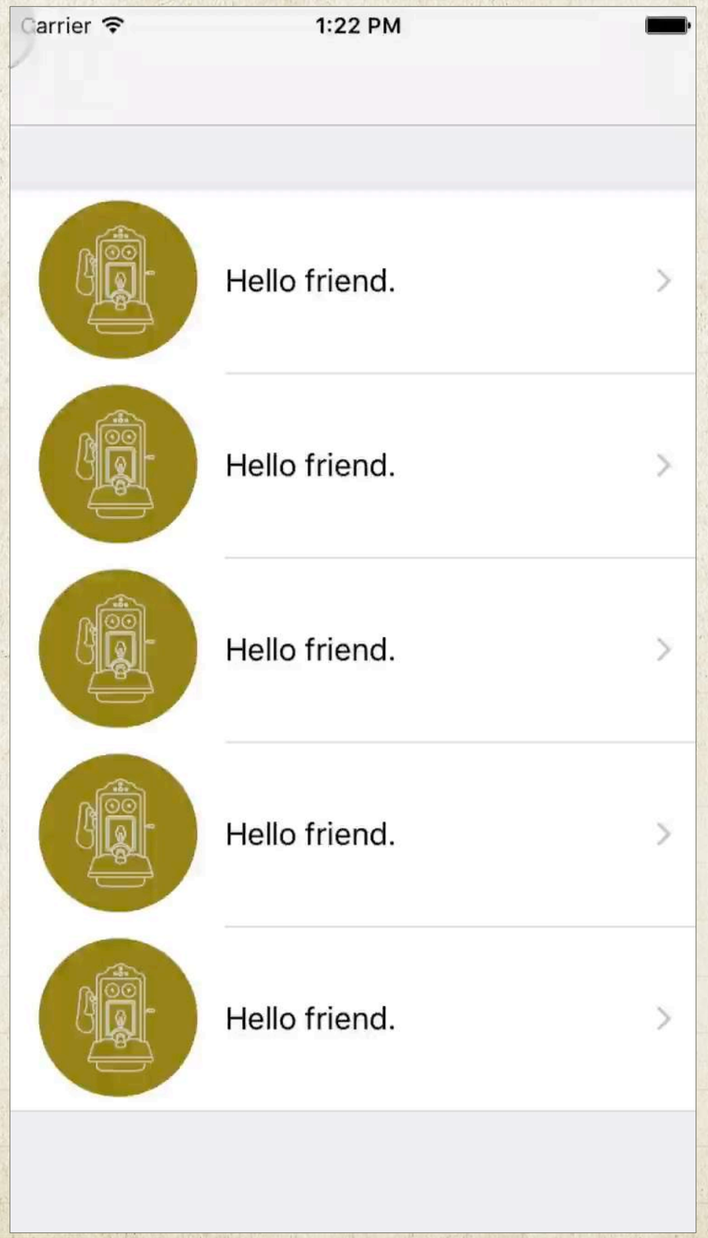
Level 5

Navigation

Section 1 - Transitioning Between View Controllers

APP EVOLUTION with Swift

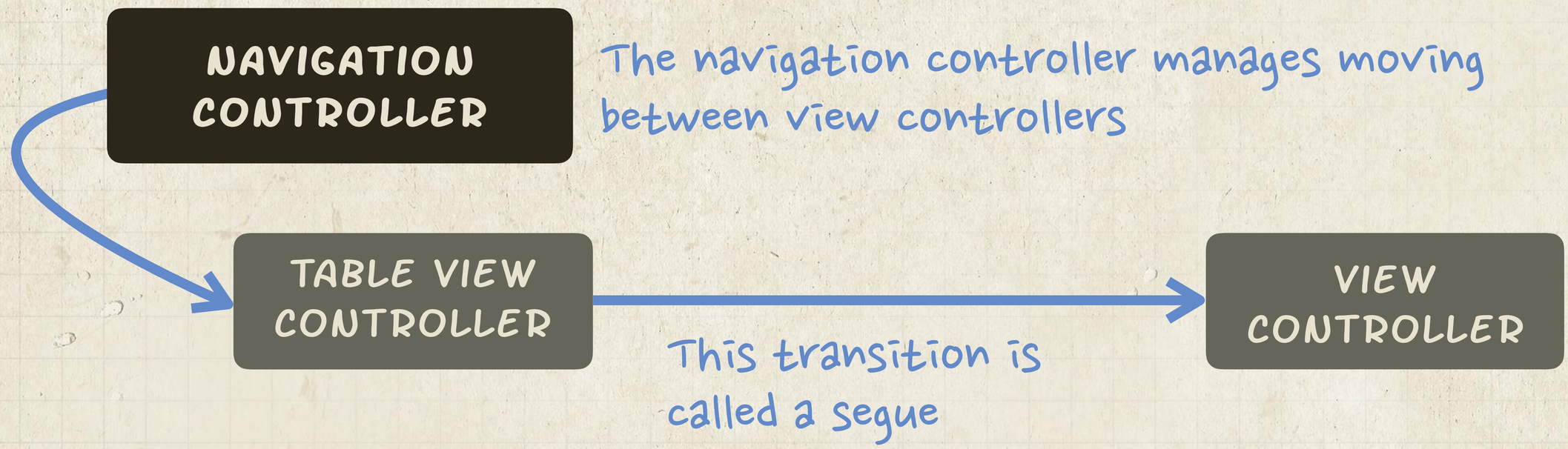
Demo: Transitioning Between Screens



APP EVOLUTION with Swift

Where Navigation Controllers Fit in the Hierarchy

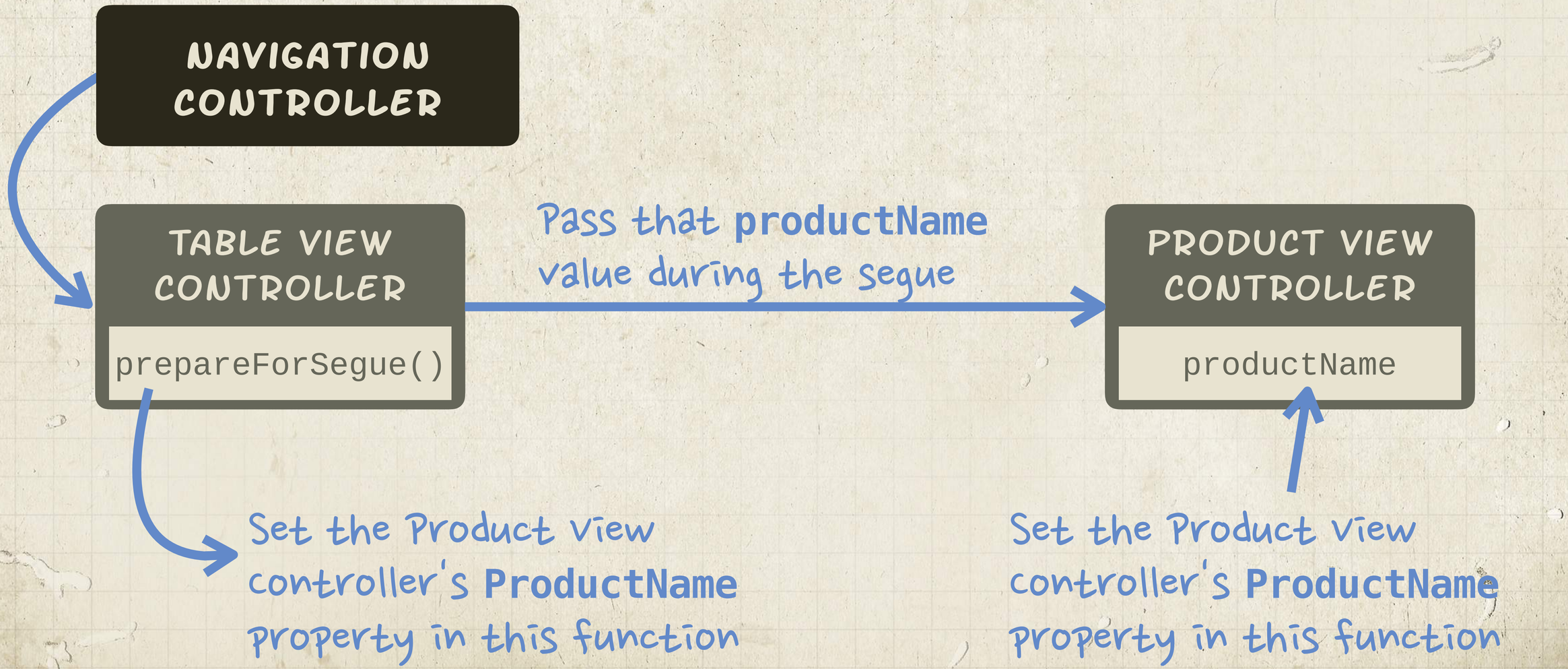
You can't switch between view controllers unless they are a part of a navigation controller stack.



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

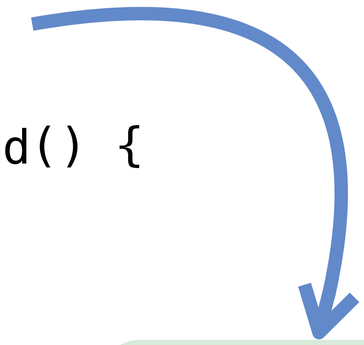
```
import UIKit

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
    }
    ...
}
```



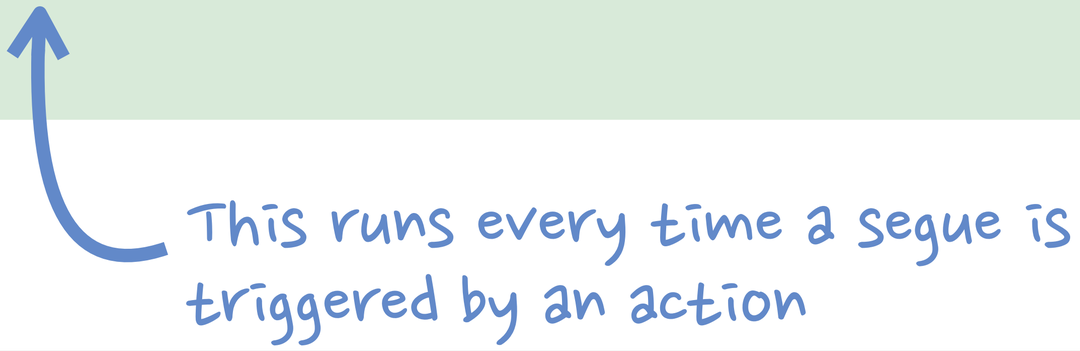
The value that's passed into this property will be displayed in the label

Adding the prepareForSegue Function

Add the prepareForSegue 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 prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {

  }

}
```

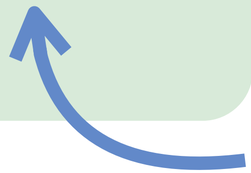


Checking for the Right Segue

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

ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
  ...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    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

`destinationViewController` *Property*

The destination view controller for the segue. (read-only)

Declaration

SWIFT

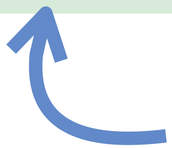
```
var destinationViewController: UIViewController { get }
```

APP EVOLUTION with Swift

Capturing the Destination in a Variable

ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
  ...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController
    }
  }

}
```



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 prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController 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 prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController 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.destinationViewController as? ProductViewController
```

Optional now

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

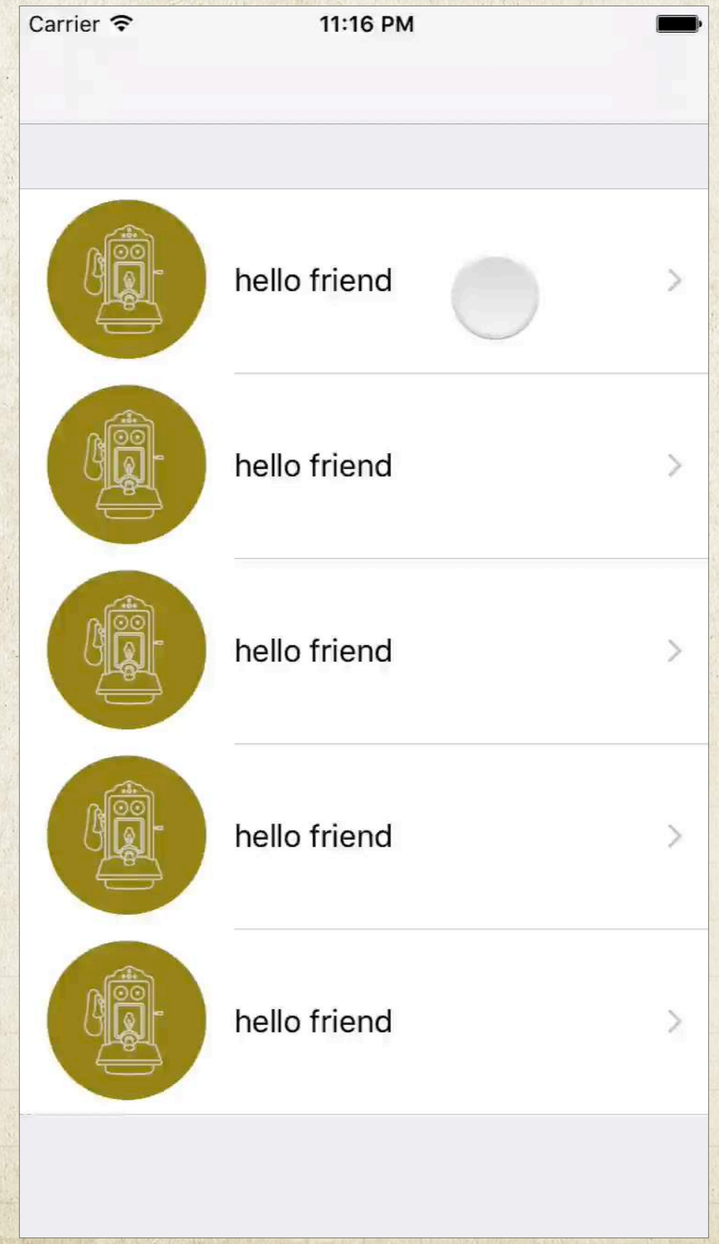
Set the Property on the ProductViewController

ProductsTableViewController.swift

```
import UIKit

class ProductsTableViewController: UITableViewController {
  ...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController 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

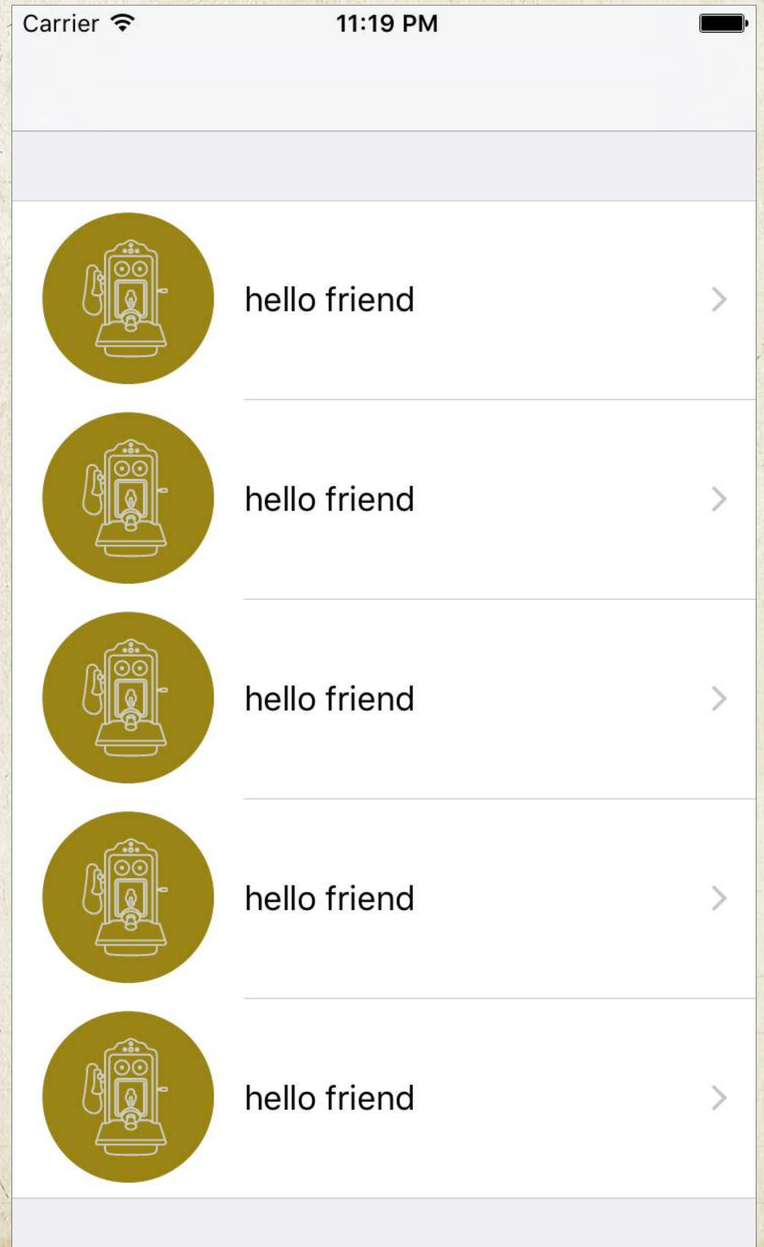
Level 5

Navigation

Section 2 - Displaying Dynamic Table Cell Data

APP EVOLUTION with Swift

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 is also optional

This array can only contain strings

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,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> 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,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> 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,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> 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,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  { ... }
  ...
}
```

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

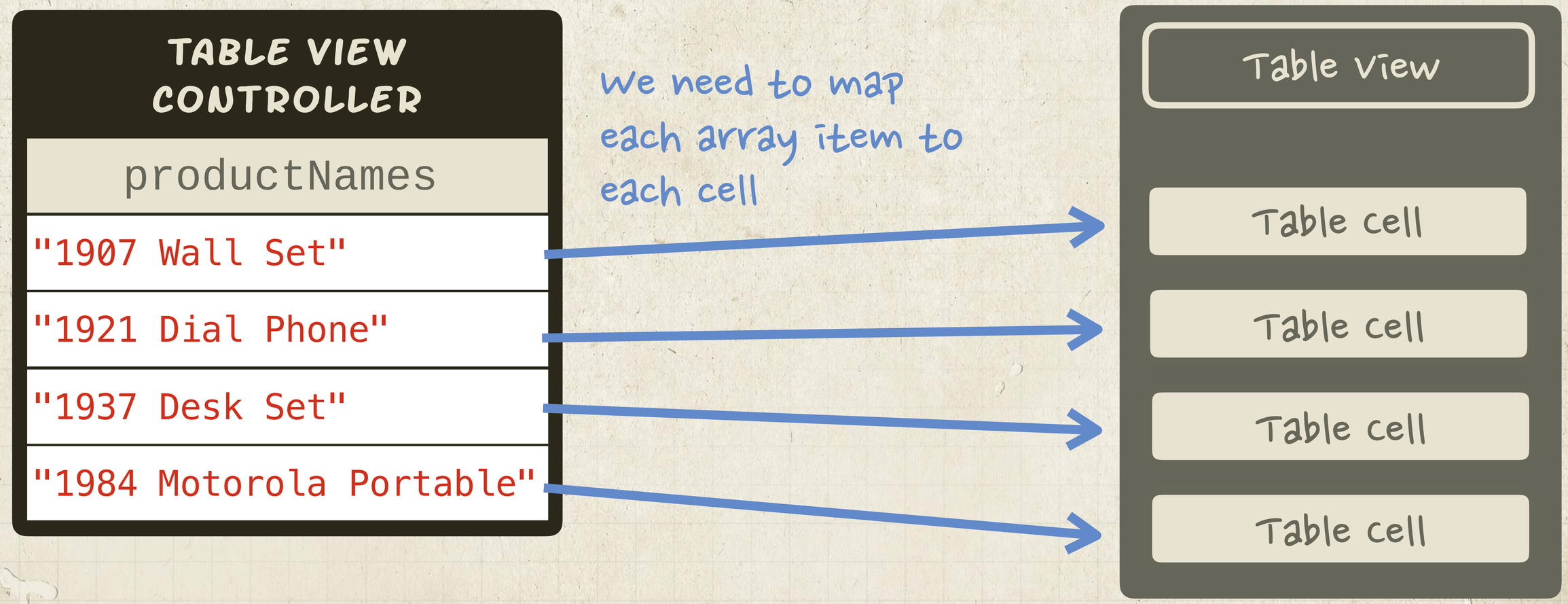
ProductsTableViewController.swift

```
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,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> 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

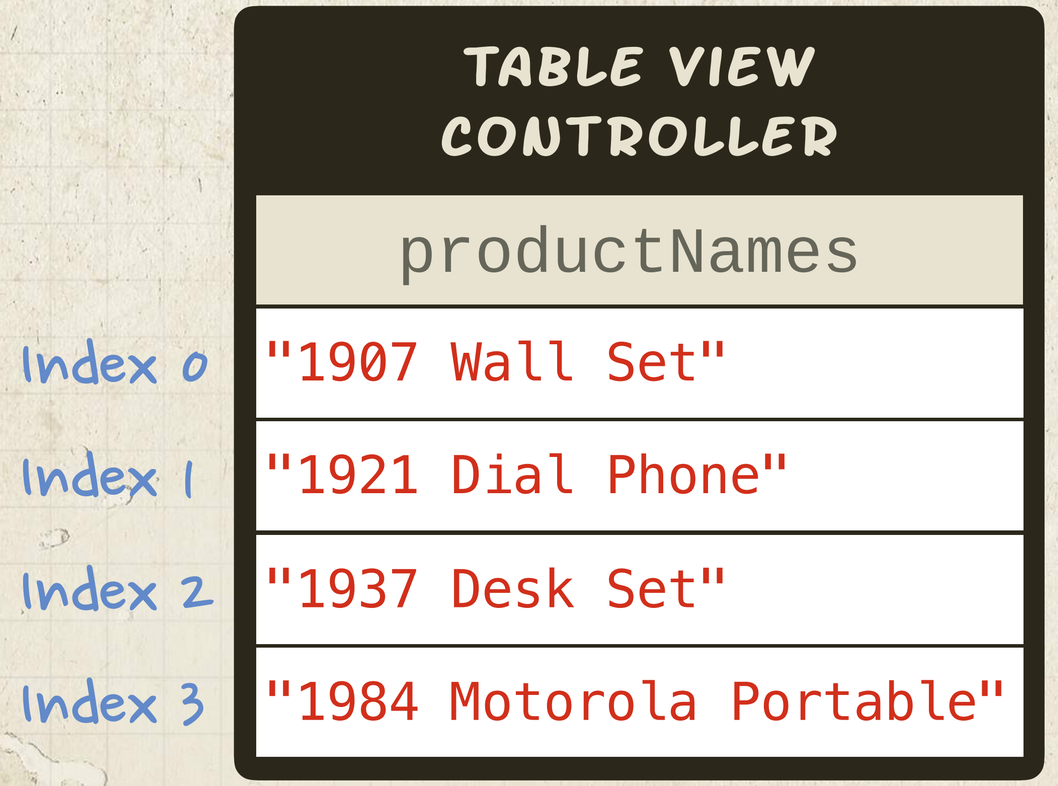
ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {

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

  override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("ProductCell", forIndexPath: 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"
```

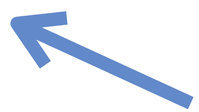
```
print(productNames[3])
```

```
"1984 Motorola Portable"
```

Plan of Attack for Updating the Cells

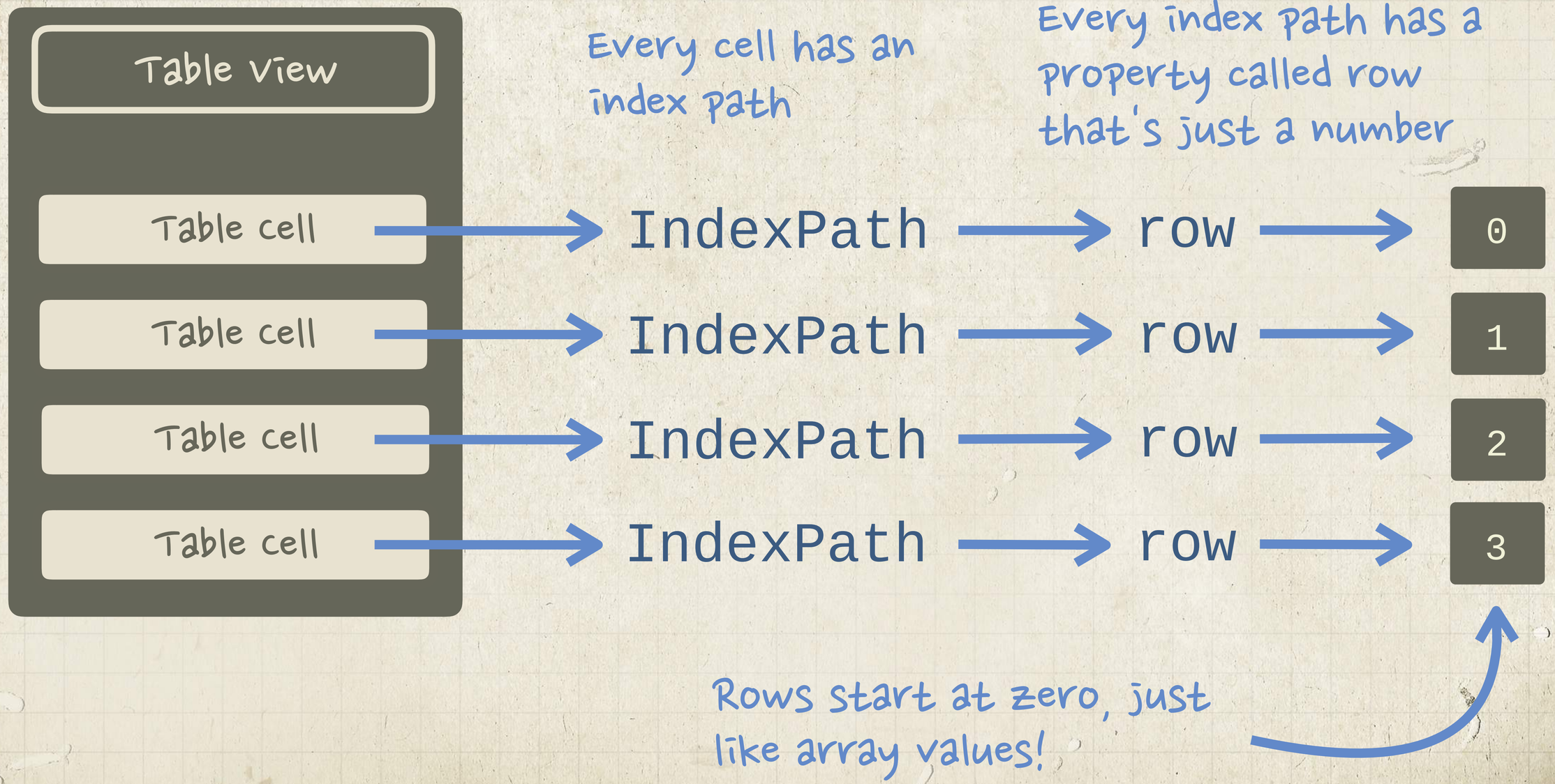
ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {

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

  override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("ProductCell", forIndexPath: indexPath)

    let productName = 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



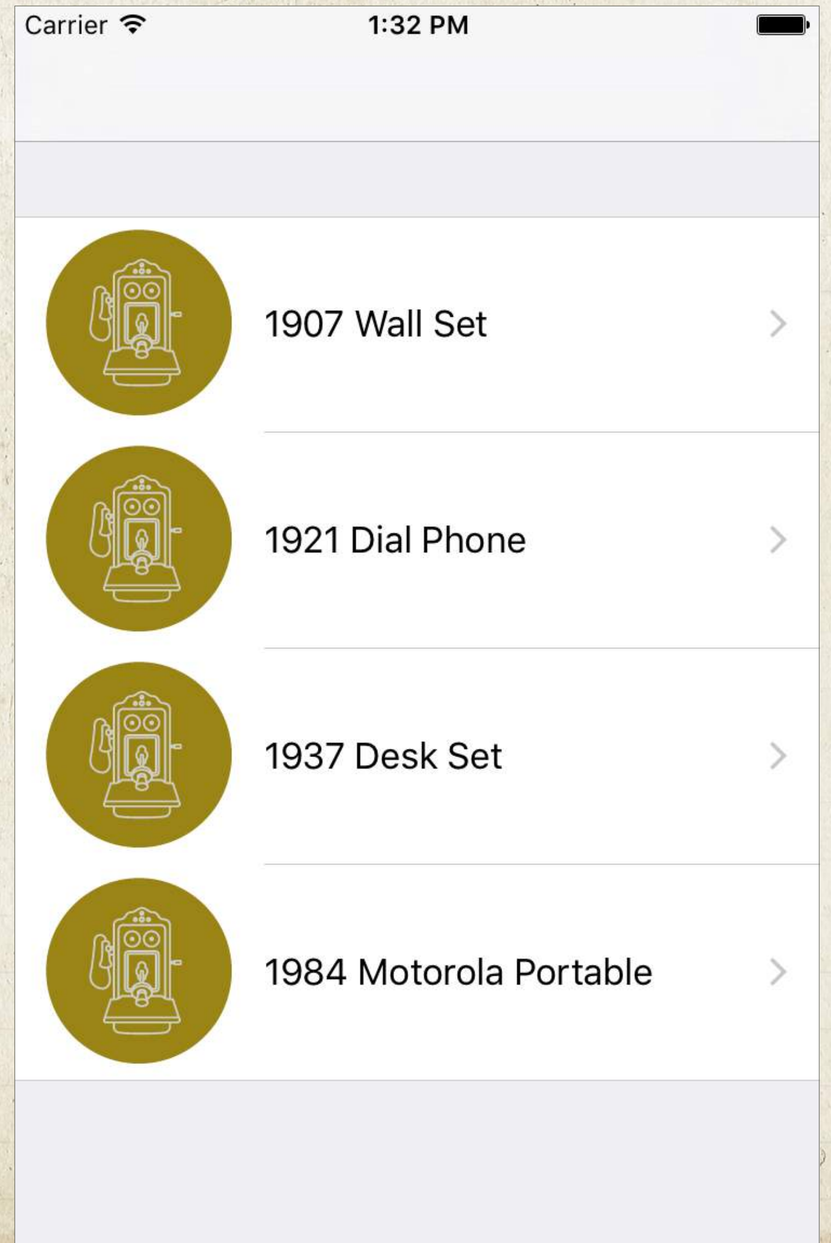
Using the indexPath to Access an Array Item

ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {

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

  override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {
    let cell = tableView.dequeueReusableCellWithIdentifier("ProductCell", forIndexPath: 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

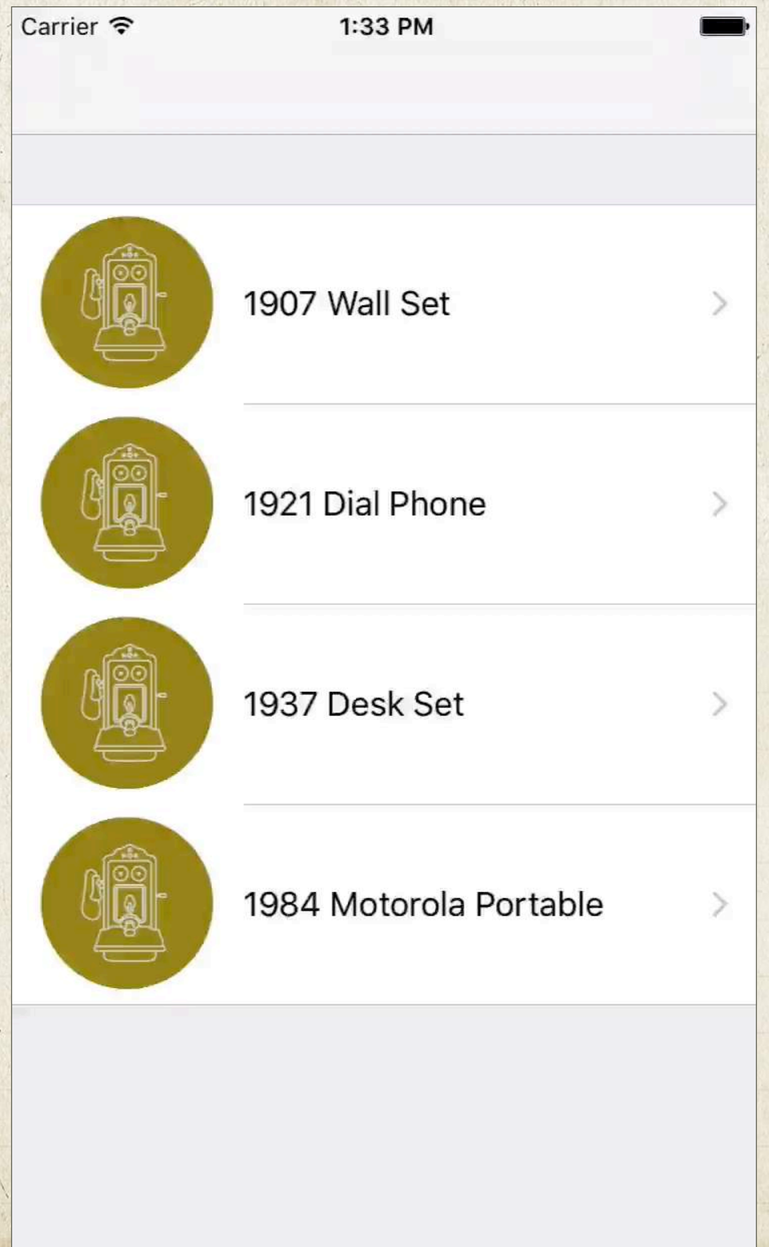
Level 5

Navigation

Section 3 - Passing Dynamic Data During a Transition

APP EVOLUTION with Swift

Problem: We Want the Product Detail to Update Too



APP EVOLUTION with Swift

The Plan for Updating prepareForSegue

ProductsTableViewController.swift

```
class ProductsTableViewController: UITableViewController {
  ...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController 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

```
...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

      let cell = sender as? UITableViewCell

    }
  }
}
```

"Sender" Here is the cell that caused the segue to happen

Let the compiler know the **sender** is a **UITableViewCell**

Using the Cell to Get an indexPath

ProductsTableViewController.swift

```
...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

      let cell = sender as? UITableViewCell
      if let c = cell {
        let indexPath = tableView.indexPathForCell(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 prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

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

The index path that's returned is also an optional

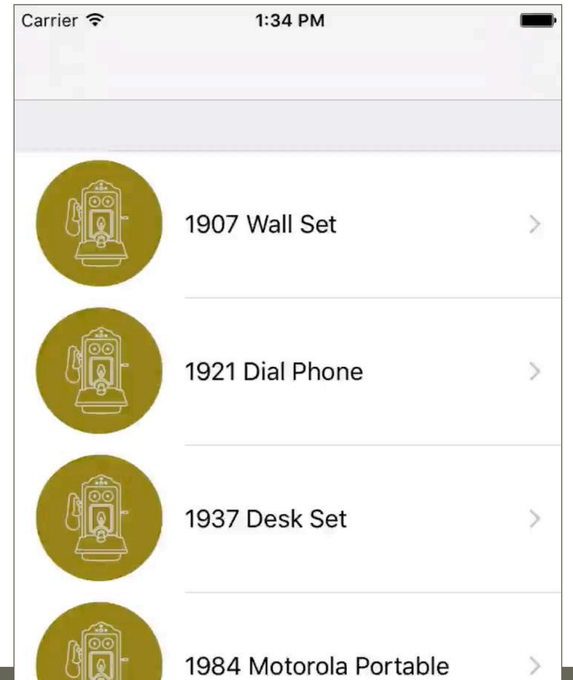
Use the index path to get the correct item from the array

Send the Name to the ProductViewController

ProductsTableViewController.swift

```
...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

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

Ambiguous variable names

Lots of code block nesting

Problem: This Is a Lot of Messy Nested Code

ProductsTableViewController.swift

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

Ambiguous variable names

Lots of code block nesting

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.indexPathForCell(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 prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

      if let cell = sender as? UITableViewCell {
        let indexPath = tableView.indexPathForCell(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 prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

      if let cell = sender as? UITableViewCell {
        if let indexPath = tableView.indexPathForCell(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.indexPathForCell(cell)
if let indexPath = indexPath {
  ...
}
```

Refactor Setting the Product Name to 1 Line

ProductsTableViewController.swift

```
...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

      if let cell = sender as? UITableViewCell {
        if let indexPath = tableView.indexPathForCell(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

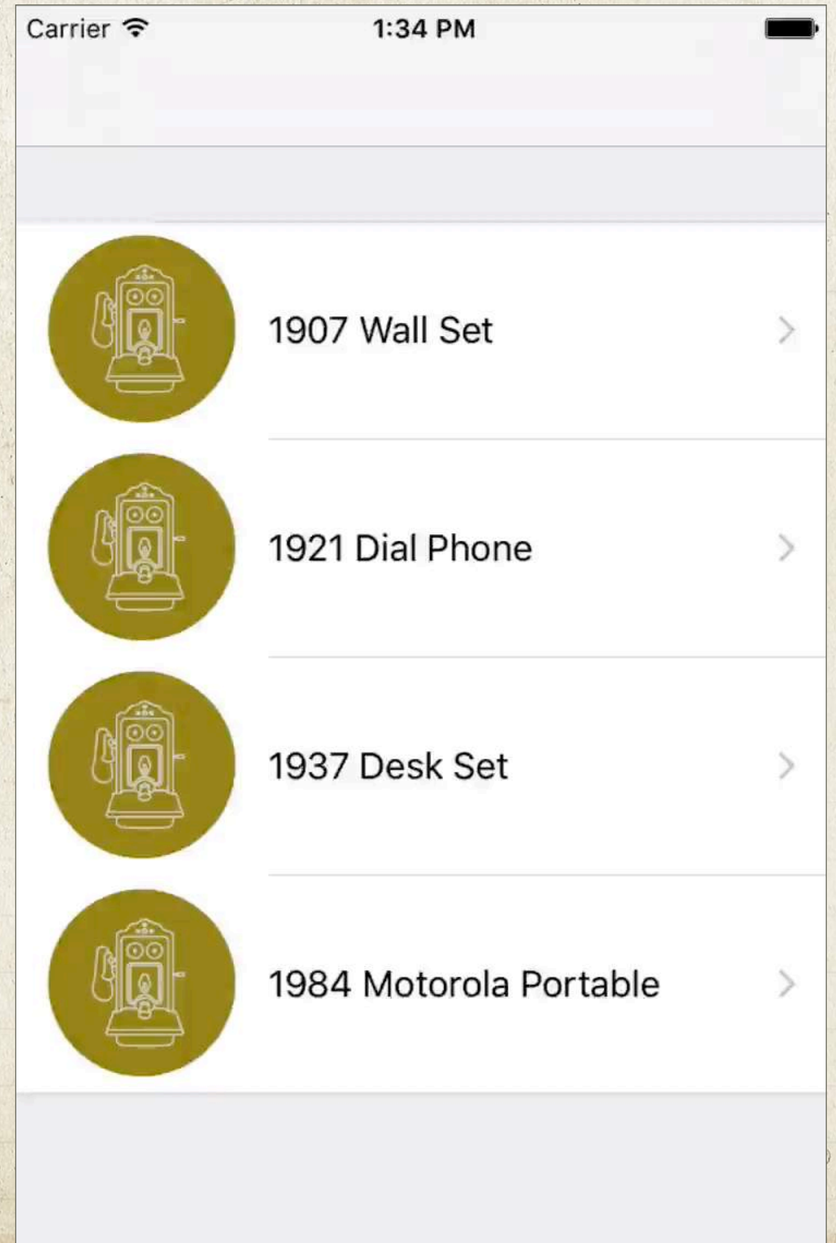
```
...

  override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "ShowProduct" {
      let productVC = segue.destinationViewController as? ProductViewController

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

stops running the function

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

Now safely use **indexPath**

Demo: Segue Passing Over Product Data



APP EVOLUTION with Swift

Level 6

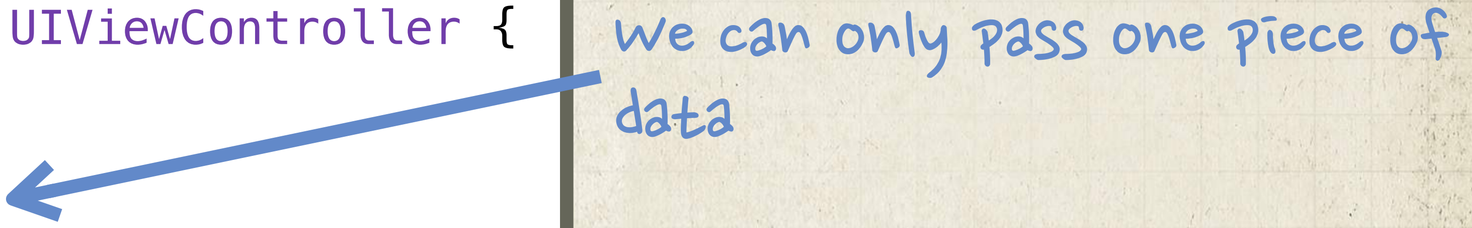
Custom Classes for Data

APP EVOLUTION with Swift

A Problem With Our Current Approach

ProductViewController.swift

```
import UIKit

class ProductViewController: UIViewController {

  ...
  var productName: String?

  override func viewDidLoad() {
    super.viewDidLoad()

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

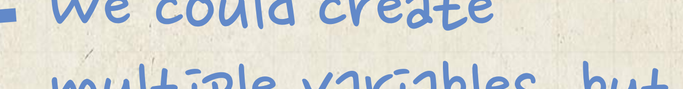


What if we wanted to pass the product and cell images?

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