UITableViews, UITableViewControllers and a Few Sage Words About Models

Mobile Computing - iOS

Custom cells section needs to be vetted

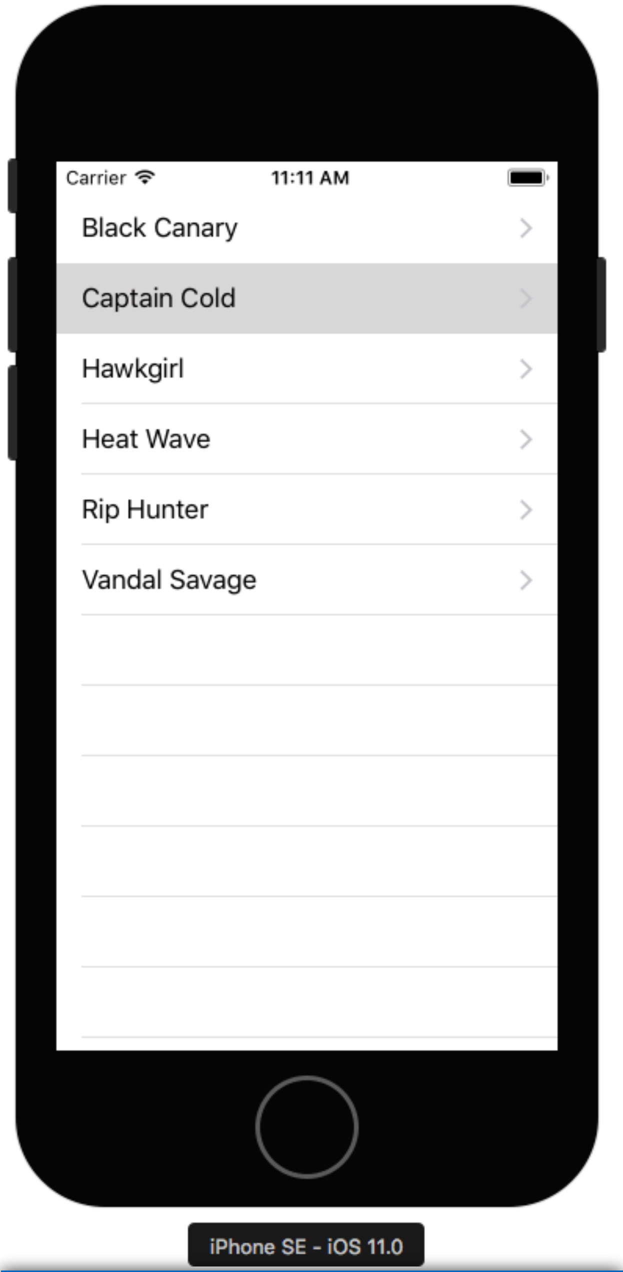
Objectives

- Students will be able to:
    - explain the purpose of table views and table view controllers
    - create apps that use table views and table view controllers
    - add image files to asset catalogs, and display images in a table view
    - explain why/how/where to create model classes
    - create and use custom cells

Overview

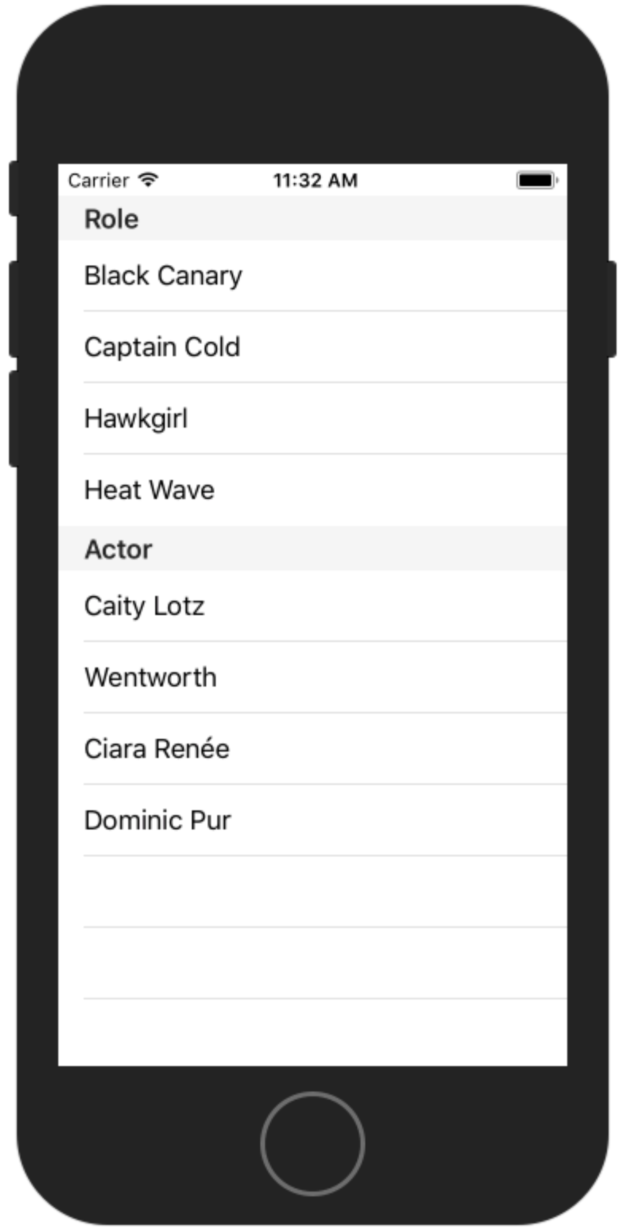
- A table is a single column, scrollable list of rows, that may be divided into sections or groups
- They are used to present information, navigate through a hierarchy of data, or select one cell from among many
- They are *ubiquitous* in iOS
- Apple's human interface guidelines has much to say on the subject

Examples

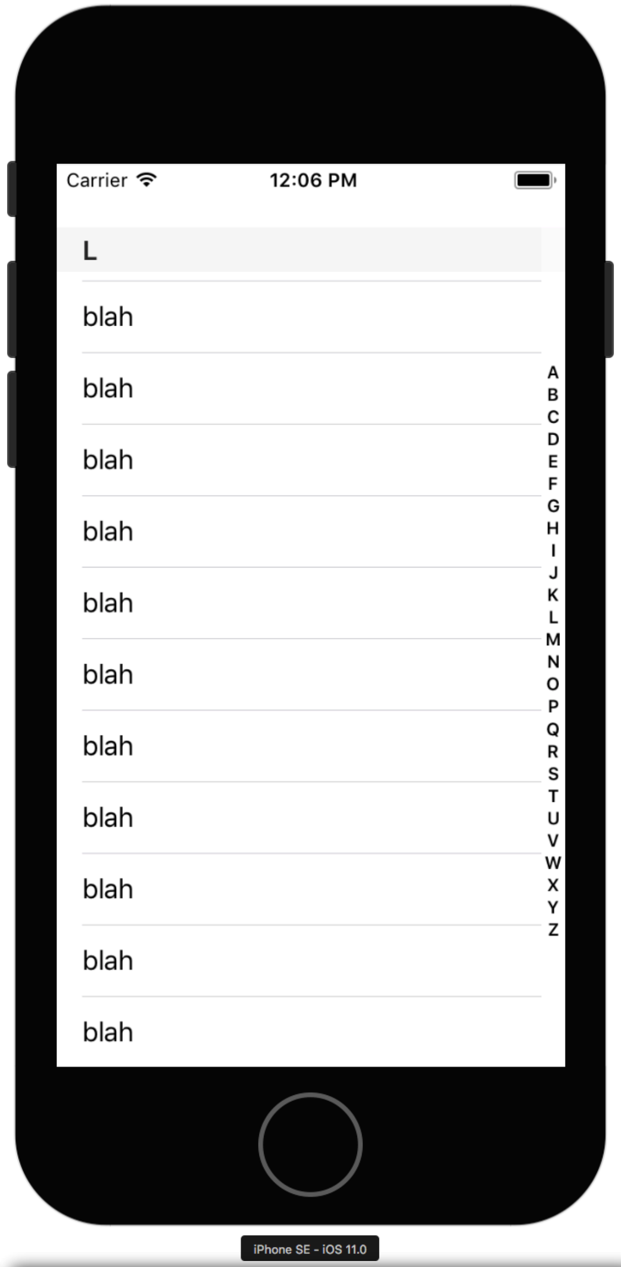

Carrier 11:11 AM

Black Canary >
Captain Cold >
Hawkgirl >
Heat Wave >
Rip Hunter >
Vandal Savage >

iPhone SE - iOS 11.0

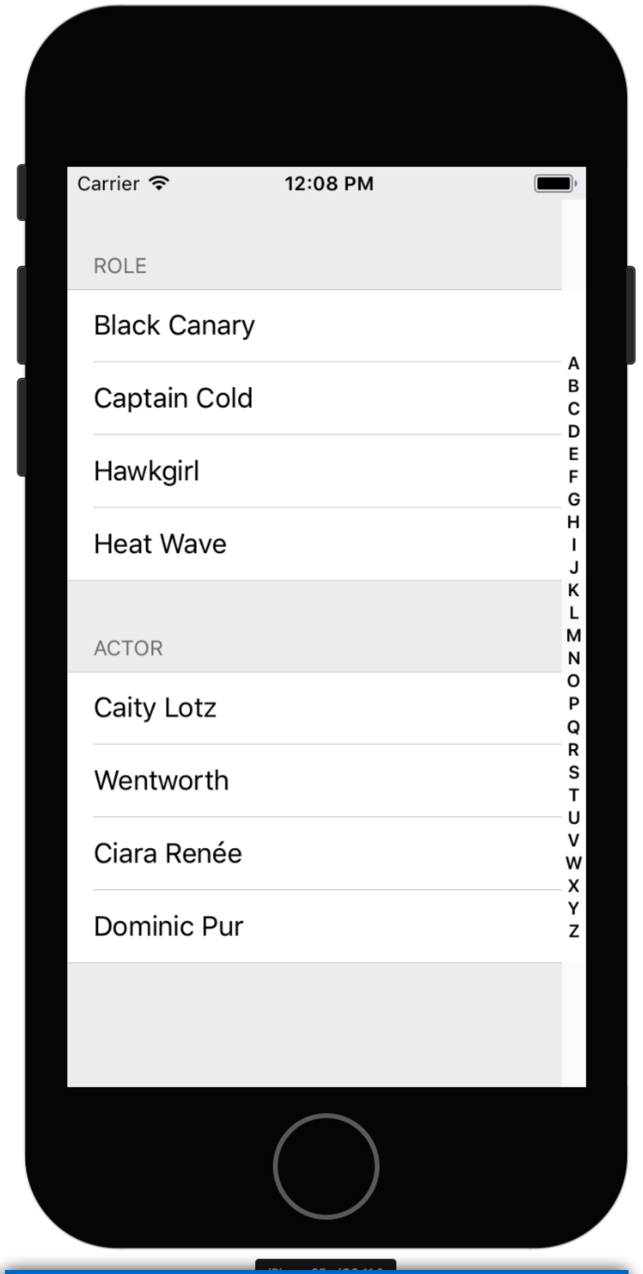

Carrier 11:32 AM

Role
Black Canary
Captain Cold
Hawkgirl
Heat Wave
Actor
Caity Lotz
Wentworth
Ciara Renée
Dominic Pur


Carrier 12:06 PM

L
blah
blah
blah
blah
blah
blah
blah
blah
blah
blah
blah

iPhone SE - iOS 11.0


Carrier 12:08 PM

ROLE
Black Canary
Captain Cold
Hawkgirl
Heat Wave
ACTOR
Caity Lotz
Wentworth
Ciara Renée
Dominic Pur

- Table style: plain
- One section
- Disclosure indicators (>)

- Two sections, titled Role and Actor. Each has a header and could have a footer
- The tableview itself can have a distinct header and footer

- Section Index (click on the letter to jump to a particular section)

- Table style: grouped — thicker headers/footers (with different typography)

Data Sources and Delegates

- A **table view** (an instance of **UITableView**), gets its data from a data source - any class that adheres to the UITableViewDataSource protocol. The data source provides 3 key pieces of information:
    - number of sections in the table view
    - number of rows per section
    - the actual cell that is displayed in each row of each section.
- A table view uses a **delegate** - any class that adheres to the UITableViewDelegate protocol - to decide what to do when a cell is selected/deselected, when an accessory is tapped, to control cell height, etc.

UITableViewDataSource

func numberOfSections(in: UITableView) -> Int
Returns the number of sections in the table view.

func tableView(UITableView, numberOfRowsInSection: Int) -> Int Returns the number of rows in a given section of a table view. **Required**.

func tableView(UITableView, cellForRowAt: IndexPath) -> UITableViewCell
Returns a cell to insert in a particular location of the table view. **Required**.

func tableView(UITableView, titleForHeaderInSection: Int) -> String?
Returns the title of the header of the specified section of the table view.

func tableView(UITableView, titleForFooterInSection: Int) -> String?
Returns the title of the footer of the specified section of the table view.

IndexPath

- A handy-dandy struct to identify each cell in a table view.
- Has properties (and other stuff)
  - **section** - Which section is the cell in
  - **row** - Which row in the section is the cell in.

Table Views and Storyboard

- Storyboard makes it easy to work with table views. There are 3 basic steps:

1. Modify a UIViewController so it implements both UITableViewDataSource and UITableViewDelegate

2. Drag in a UITableView into your scene and configure it

3. Control drag from the UITableView to the view controller, making it serve as both the delegate and data source

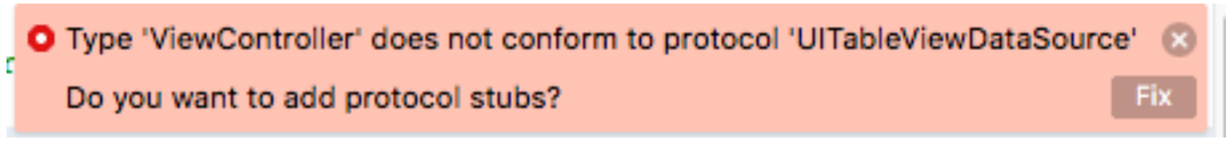
ICE: Using Storyboard

1. Create a single view app, **Best Restaurants**

2. Refactor* ViewController as RestaurantsViewController, and make it adhere to both protocols:

```
class RestaurantsViewController :
UIViewController, UITableViewDelegate,
UITableViewDataSource
```

3. When prompted, add protocol stubs 😍



Type 'ViewController' does not conform to protocol 'UITableViewDataSource'

Do you want to add protocol stubs?

*use the menu: if you manually rename the class, that won't change it in Storyboard

ICE: Using Storyboard

4. Drag a Table View (component) into the view, and position it so it completely fills the Safe Area

5. To configure the UITableView, select it and do this:

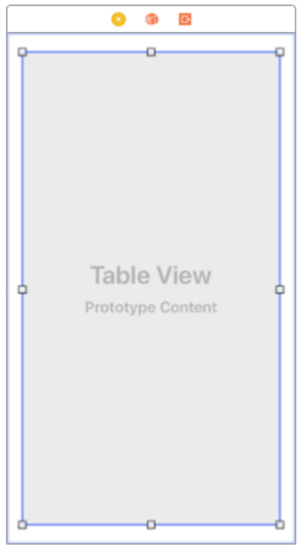
   1. Under constraints issues, reset to suggested constraints

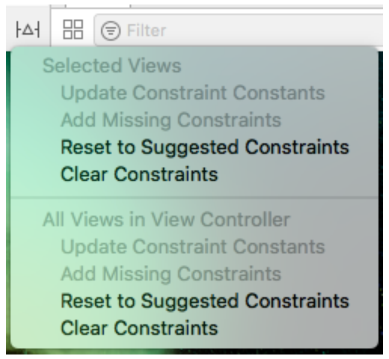
   2. Control drag to the View Controller and choose data source; repeat the process with delegate

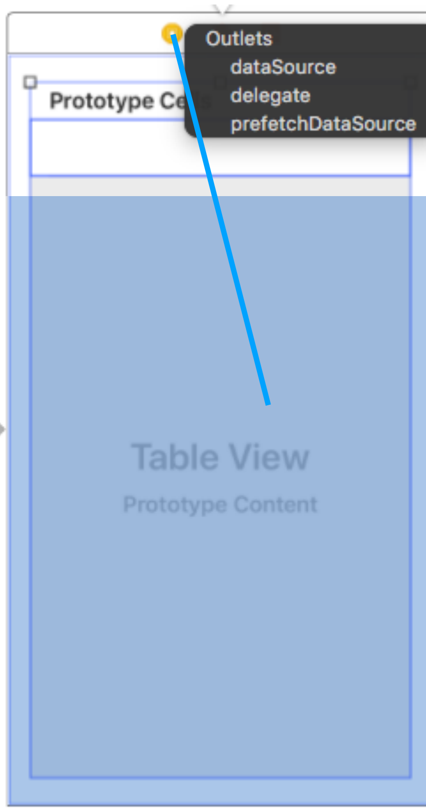
   3. Set the number of prototype cells (under attributes) to 1

   4. Select the prototype cell and set its style to Basic and its identifier to **restaurantCell**

6. Complete the protocol stubs in RestaurantsViewController as shown on the next slide, then run the app to enjoy the fruits of your labor 😍




Filter
Selected Views
Update Constraint Constants
Add Missing Constraints
Reset to Suggested Constraints
Clear Constraints
All Views in View Controller
Update Constraint Constants
Add Missing Constraints
Reset to Suggested Constraints
Clear Constraints



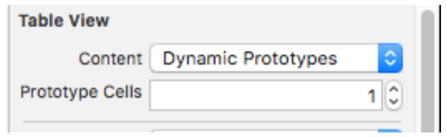


Table View

Content: Dynamic Prototypes

Prototype Cells: 1

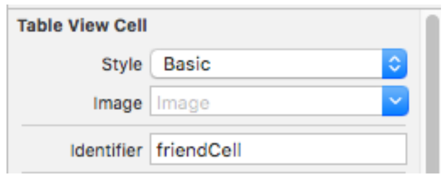


Table View Cell

Style: Basic

Image: Image

Identifier: friendCell

ICE: Using Storyboard

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

  var restaurants:[String] = ["Planet Sub", "Subway", "Applebee's",
                              "Pizza Ranch", "A&Gs", "Jimmy Johns"]

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

 func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
          -> UITableViewCell {
     let cell = tableView.dequeueReusableCell(withIdentifier: "restaurantCell")
     cell?.textLabel?.text = restaurants[indexPath.row]
     return cell!
  }

// the rest of the View Controller is as provided in the Xcode template
```

Dequeuing Reusable Cells

- A table view shows just a few cells at a time: so rather than instantiate a separate cell for each row (there could be 000's), cells not currently visible are stored in an internal queue, then fetched as needed with **dequeueReusableCell(withIdentifier:)**.

- Since a table view can store different kinds of cells, each cell type get a **reuse identifier**. To handle multiple cell types, we would:

    1. Create as many prototype cells as necessary (in the table view's attributes inspector)

    2. Configure each prototype cell as needed, and give it a unique reuse identifier

    3. Use dequeueReusableCell(withIdentifier:) to retrieve whichever is needed

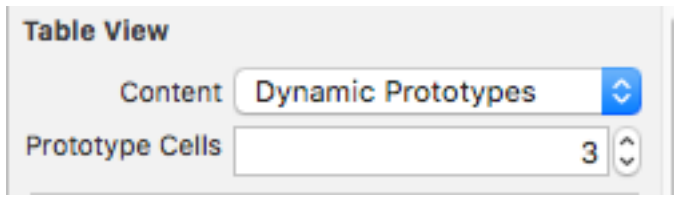


Table View
Content: Dynamic Prototypes
Prototype Cells: 3

What Storyboard Just Did For Us

```swift
override func loadView(){
  let viewToBe:UIView = UIView()                          // every view controller needs a view ... we make it here
  let tableView:UITableView = UITableView(frame: CGRect.zero, style: .plain) // now we have a table view

  tableView.delegate = self // our tableView will use this class for both its delegate and data source
  tableView.dataSource = self

  // when asked for a cell with reuse identifier of "restaurantCell", instantiate a UITableViewCell object
  // Replace UITableViewCell with subclass of UITableViewCell if you need customization
  tableView.register(UITableViewCell.self, forCellReuseIdentifier: "restaurantCell")
  viewToBe.addSubview(tableView)
  self.view = viewToBe // now our UIViewController has the view it needs

  let bigLabel:UILabel = UILabel()
  bigLabel.text = "A Simple Example"
  bigLabel.font = UIFont(name: "Helvetica", size: 24.0)
  bigLabel.textAlignment = .center
  bigLabel.backgroundColor = UIColor.white
  self.view.addSubview(bigLabel)

  // OK, constraint time! the view's safeAreaLayoutGuide represents the Safe Area, that will be visible regardless
  // of status bar or navigation bar (on top) or tab bar (bottom)
  bigLabel.translatesAutoresizingMaskIntoConstraints = false
  bigLabel.leadingAnchor.constraint(equalTo:view.safeAreaLayoutGuide.leadingAnchor , constant: 0.0).isActive = true
  bigLabel.trailingAnchor.constraint(equalTo:view.safeAreaLayoutGuide.trailingAnchor , constant: 0.0).isActive = true
  bigLabel.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 0.0).isActive = true
  bigLabel.heightAnchor.constraint(equalToConstant: 30.0).isActive = true
  bigLabel.widthAnchor.constraint(greaterThanOrEqualToConstant: 100.0).isActive = true

  tableView.translatesAutoresizingMaskIntoConstraints = false
  tableView.leadingAnchor.constraint(equalTo:view.safeAreaLayoutGuide.leadingAnchor , constant: 0.0).isActive = true
  tableView.trailingAnchor.constraint(equalTo:view.safeAreaLayoutGuide.trailingAnchor , constant: 0.0).isActive = true
  tableView.topAnchor.constraint(equalTo: bigLabel.bottomAnchor, constant: 0.0).isActive = true
  tableView.bottomAnchor.constraint(equalTo: view.safeAreaLayoutGuide.bottomAnchor, constant: 0.0).isActive = true
}
```

Setting the Delegate/DataSource in IB and Code: Which is Faster?

- A table view can use *any* class that adheres to the DataSource and Delegate protocols.

- When working in Interface Builder, it is convenient to use the view controller that houses the table view, because control-drag makes it joyously easy to establish the delegate and data sources

- This is the equivalent, in code:

```
tableView.delegate = self
tableView.datasource = self
```

- Q: As a developer, which is faster? 🐌🐇 Which do you find more appealing? 🍌🙄

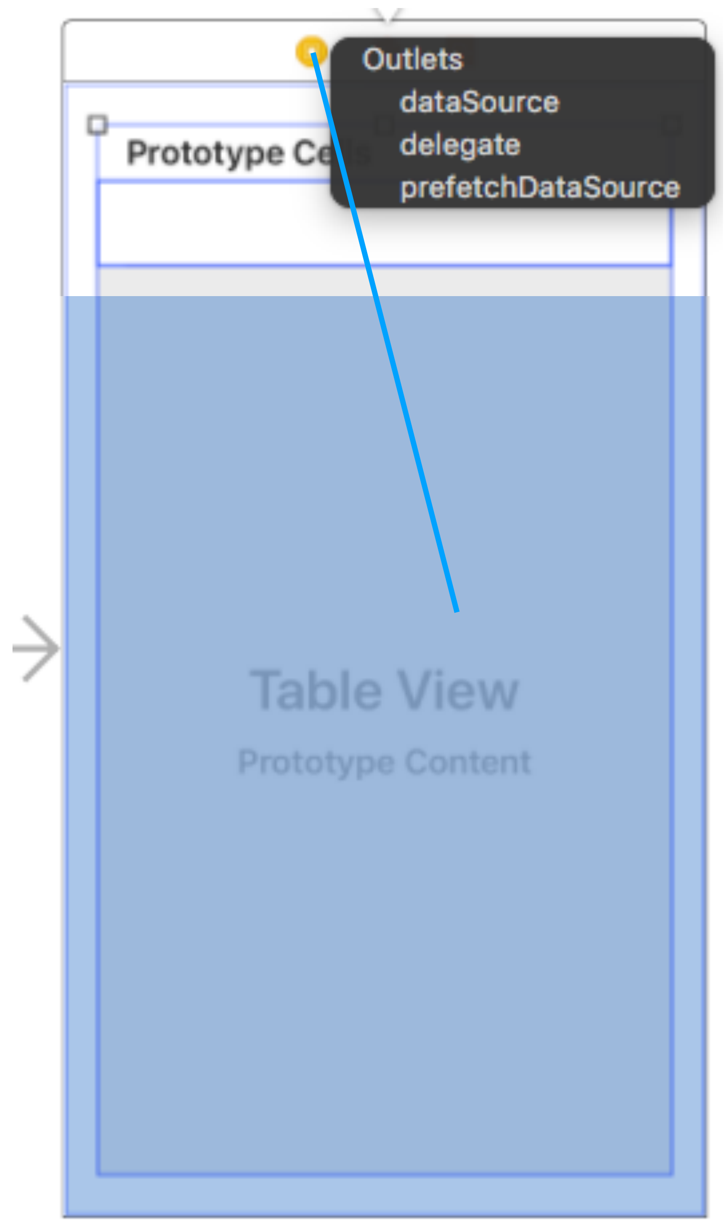


Table View Controllers 🎁

- In all of our examples thus far, we have had to
  1. instantiate a UITableView (either in code or by dragging one into a storyboard)
  2. make a class adhere to the UITableViewDelegate and UITableViewDataSource protocols
  3. designate that class as the data source and delegate for the UITableView
- This has to be done every time we need a table view, i.e., this is *boiler plate code*.
- To avoid this, iOS defines a <u>UITableViewController</u>

And now for something completely different ... sort of!

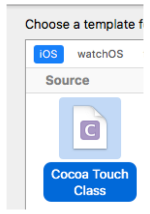
Table View Controllers

- A table view controller (a UITableViewController instance) is a specialized view controller that is perfect when your UI is *just* a table view. It automatically:
  1. generates a view consisting of a table view, sized to fit the entire safe area
  2. anoints itself as that table view's data source and delegate
- The above implies the UITableViewController's declaration:

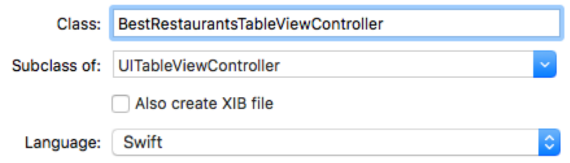
  class UITableViewController : UIViewController, UITableViewDataSource, UITableViewDelegate

ICE: UITableViewControllers in Storyboard

Code Steps:



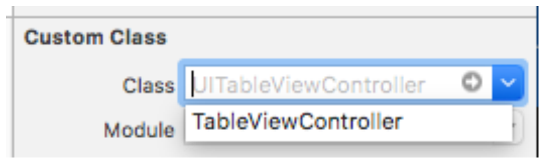
1. Create a new file (⌘N), a UITableViewController subclass, **RestaurantsTableViewController (BFTVC)**


Class: BestRestaurantsTableViewController
Subclass of: UITableViewController
Also create XIB file
Language: Swift

2. Copy the restaurants array from RestaurantsViewController (BFVC) into BFTVC, so we have a data source
3. Copy, from BFVC, the stub methods returning # of rows and table view cells. When you are done it should include the methods on the next slide.

Storyboard Steps:

4. Drag a UITableViewController into storyboard
5. Change its identity from UITableViewController to **RestaurantsTableViewController**


Custom Class
Class: UITableViewController
Module: TableViewController

6. Give it a single prototype cell a **basic** style and reuse identifier of **restaurantCell**

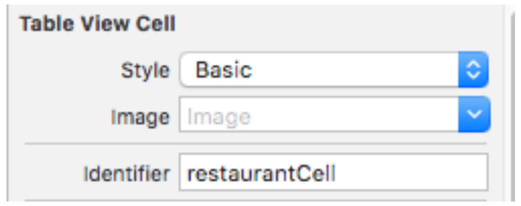
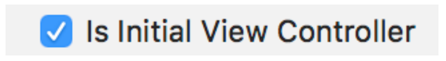
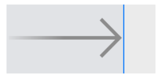


Table View Cell
Style: Basic
Image: Image
Identifier: restaurantCell

7. Make it the Initial View Controller


Is Initial View Controller



8. Delete RestaurantsViewController from storyboard (to avoid a warning about it being unreachable)

Retrofitting Best Restaurants with a TVC: fun for the whole family! 😉

RestaurantsTableViewController

```
class RestaurantsTableViewController: UITableViewController {

    var restaurants:[String] = ["Planet Sub", "Subway", "Applebee's",
                                "Pizza Ranch", "A&Gs", "Jimmy Johns" ]
    override func viewDidLoad() {
        super.viewDidLoad()

    }

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

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

   override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
                -> UITableViewCell {
     let cell = tableView.dequeueReusableCell(withIdentifier: "restaurantCell")
     cell?.textLabel?.text = restaurants[indexPath.row]
     return cell!
    }

// ... etc.
```

Instantiating In Code

```
func application(_ application: UIApplication,
     didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?)
              -> Bool {
 // Override point for customization after application launch.
 window = UIWindow(frame: UIScreen.main.bounds) // every app belongs in a window -
                                                // should be same size as main screen
 let myTVC = TableViewController(style:UITableViewStyle.plain) // we have a table view controller
 myTVC.tableView.register(UITableViewCell.self, forCellReuseIdentifier: "restaurantCell")
                              // type of cell to be recycled when "cell" is asked for
 window?.rootViewController = myTVC // make that TVC the window's root view controller

 window?.makeKeyAndVisible() // let's see it!
 return true

 }
```

Table View Cell Details

- A table view cell (an instance of UITableViewCell) can display text and an image, the exact appearance depending upon its selected style

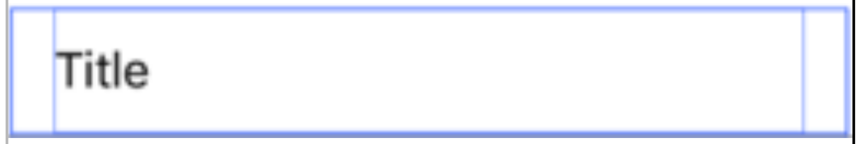
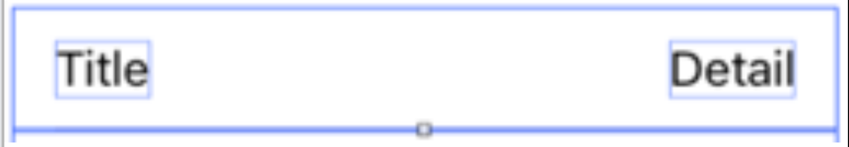
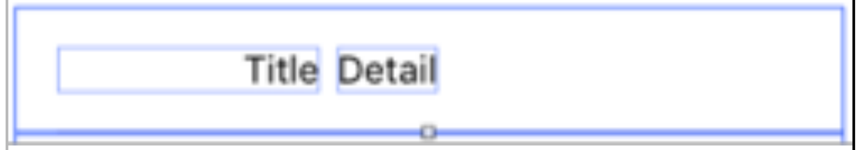
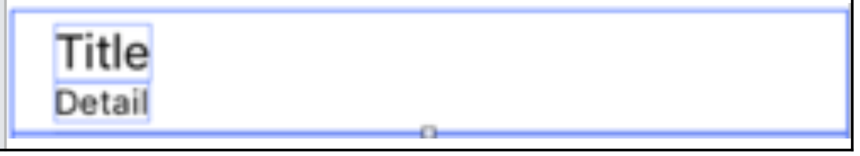
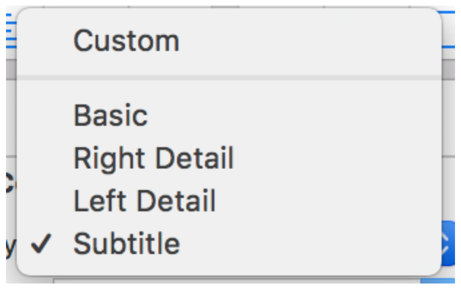
Style Name (IB)	Style Name (enum)	Example
Basic	.default	
Right Detail	.value1	
Left Detail	.value2	
Subtitle	.subtitle	

Table View Cell Details

- Basic is the default. All styles except left detail can display an image on the left; all can display an accessory image to the right.
- The title, detail, and image are accessed via a UITableViewCell's **textLabel**, **detailTextLabel**, and **imageView** properties, respectively. These are all optionals, so unwrap before using.
- Styles can be either specified via the inspector in storyboard or programmatically (with enums)



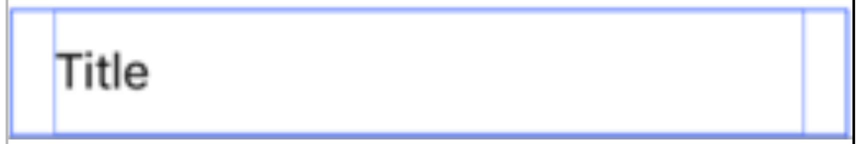
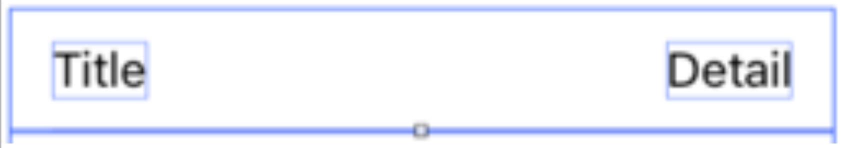
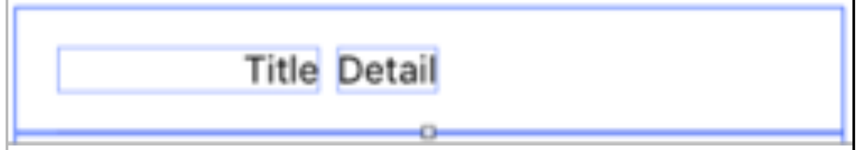
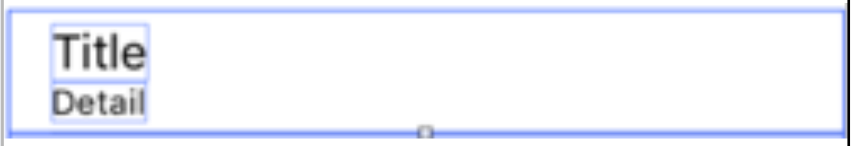
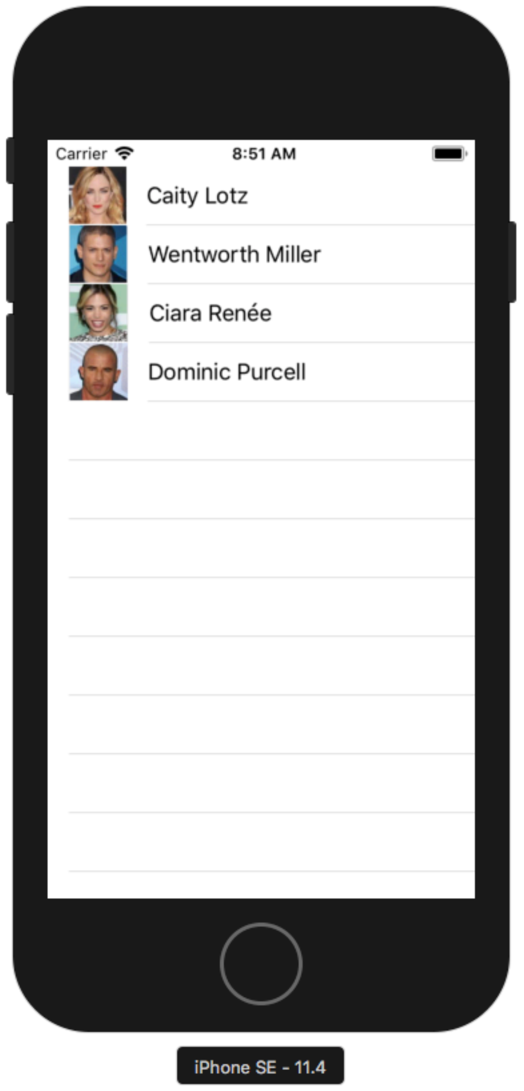
Style Name (IB)	Style Name (enum)	Example
Basic	.default	
Right Detail	.value1	
Left Detail	.value2	
Subtitle	.subtitle	

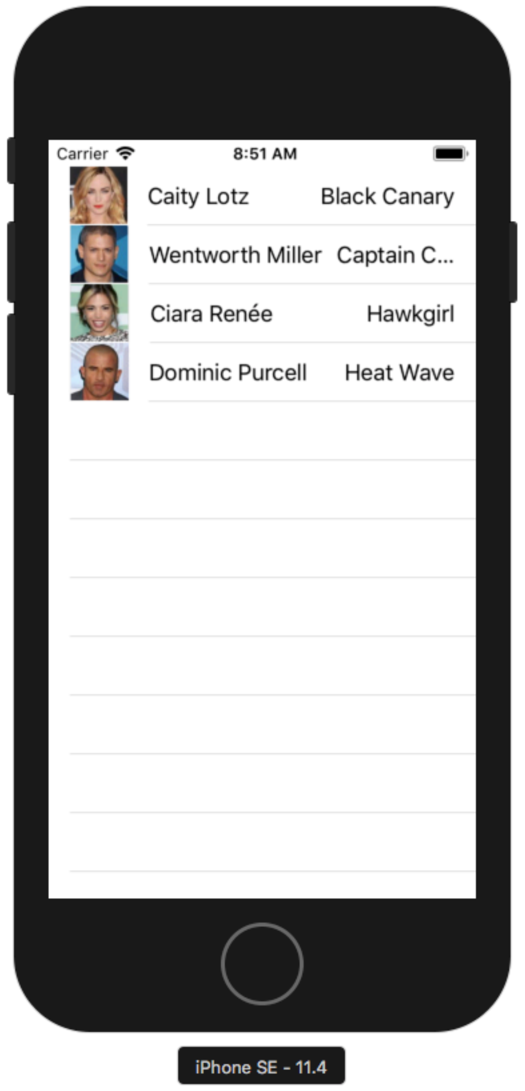
Table View Cell Details - An Example


Carrier 8:51 AM

Caity Lotz
Wentworth Miller
Ciara Renée
Dominic Purcell

iPhone SE - 11.4

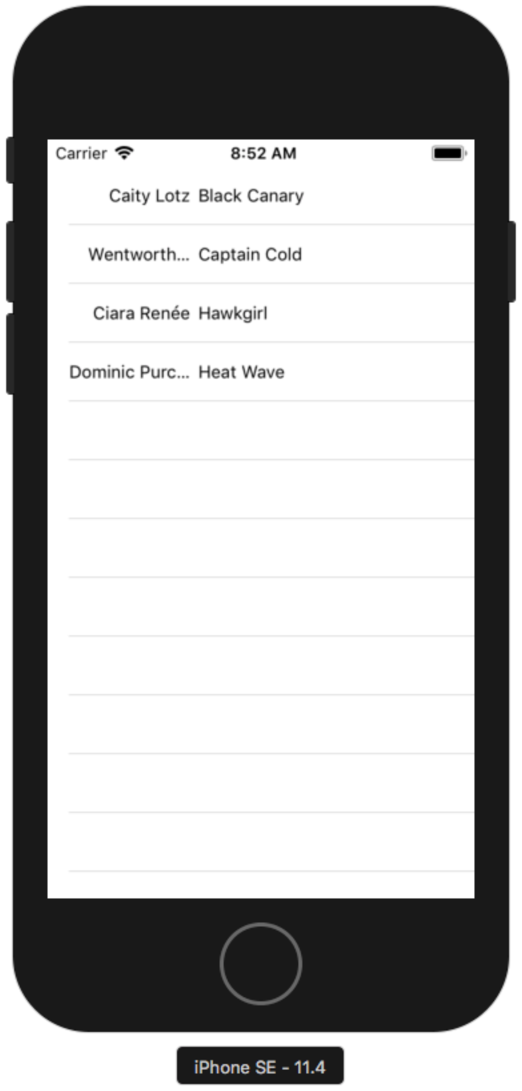
Basic


Carrier 8:51 AM

Caity Lotz Black Canary
Wentworth Miller Captain C...
Ciara Renée Hawkgirl
Dominic Purcell Heat Wave

iPhone SE - 11.4

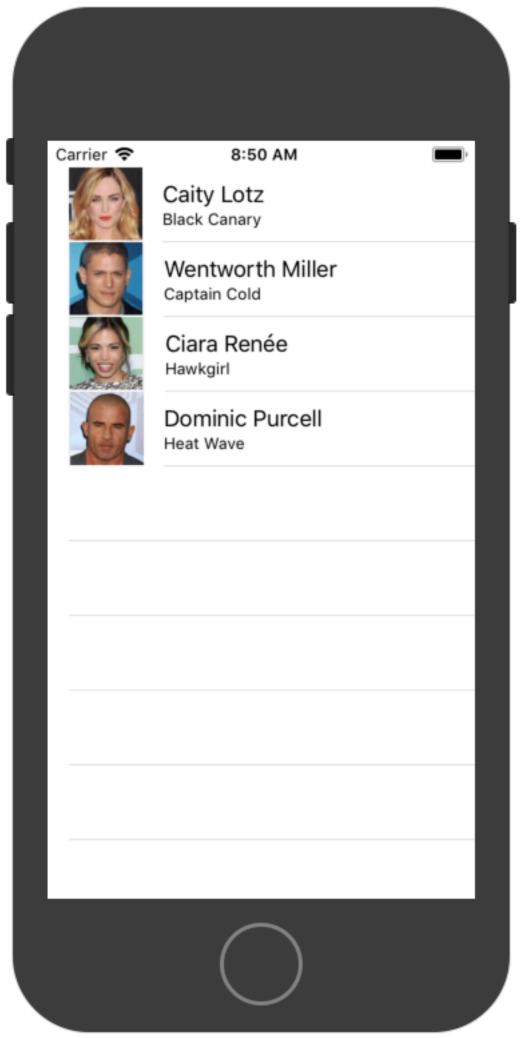
Right Detail


Carrier 8:52 AM

Caity Lotz Black Canary
Wentworth... Captain Cold
Ciara Renée Hawkgirl
Dominic Purc... Heat Wave

iPhone SE - 11.4

Left Detail

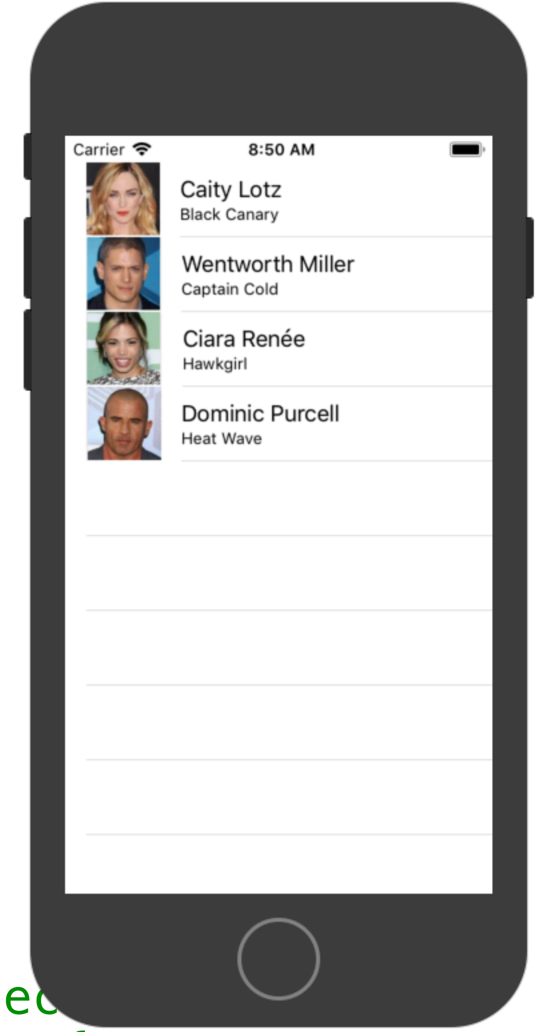

Carrier 8:50 AM

Caity Lotz
Black Canary
Wentworth Miller
Captain Cold
Ciara Renée
Hawkgirl
Dominic Purcell
Heat Wave

Subtitle

Table View Cell Details - An Example

```
struct Actor {
    let role:String
    let name:String

}


class SBViewController: UITableViewController {

 var actors:[Actor] = [  Actor(role: "Black Canary", name: "Caity Lotz"),
                         Actor(role: "Captain Cold", name: "Wentworth"),
                         Actor(role: "Hawkgirl", name: "Ciara Renée"),
                         Actor(role: "Heat Wave", name: "Dominic Pur") ]

  override func tableView(_ tableView: UITableView, numberOfRowsInSection sec
                                                                       -> Int {
        return legends.count // this is the total number of cells
    }


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

        let cell = tableView.dequeueReusableCell(withIdentifier: "cell")!
        cell.textLabel?.text = actors[indexPath.row].name
        cell.detailTextLabel?.text = actors[indexPath.row].role
        cell.imageView?.image = UIImage(named: "\(actors[indexPath.row].name).png")

        return cell
    }
}
```



Caity Lotz
Black Canary
Wentworth Miller
Captain Cold
Ciara Renée
Hawkgirl
Dominic Purcell
Heat Wave

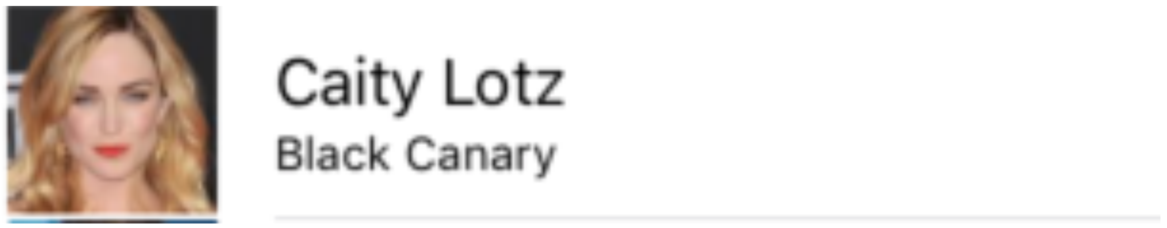
The technique shown here for creating models is quick, but we must do a better job of separating the VC and model. We address this 3 slides hence.

When using .png files, .png may be omitted: all other file formats require it

Fun Facts About Images

- In iOS, images are instances of **UIImage**, and displayed in **UIImageViews**.
- Our favorite UIImage initializer, **init?(named:)**, accepts a String, and returns an optional (why?)
- UIImageViews have an **image** property: assign it a UIImage to make it visible
- UITableViewCells have UIImageViews available for use

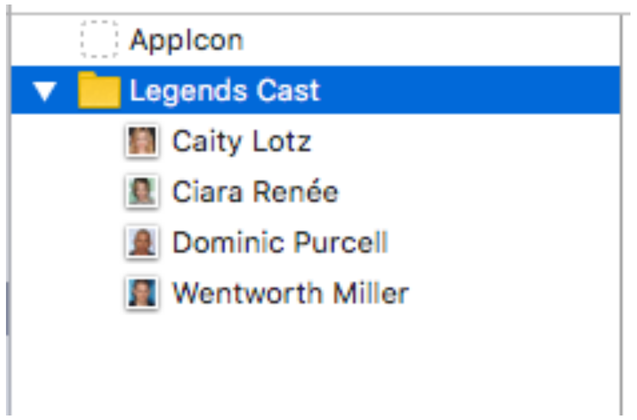
```
cell.imageView?.image = UIImage(named: "Caity Lotz.png")
```



Caity Lotz
Black Canary

Images and Asset Catalogs

- Images are most easily handled by storing them in an **asset catalog**, a storage scheme that can accommodate a multitude of asset types (icons, images, sprites, stickers, textures, watch complications, to name a few)
- Xcode provides a default asset catalog, Assets, when you make a new project
- Drag images into an asset folder to make them part of the project (be sure and **copy** them, otherwise only a reference to them will be stored, and the image will not appear 🙁)
- Images may be organized into folders within an asset catalog: ignore the folders when using init?(named:)



A Remark About Models

- You will have noticed that the model and view controller are basically one and the same -- the view control contains, and has full and unfettered access to the model. We
- Generally speaking this is a **Bad Idea***, unsustainable if the model is of any complexity (e.g., what if we had both actors and writers?). We need better separation of duties among model, view and controller.
- The next slide shows our first pass at solving this problem: a separate struct, in its own file, that encapsulates the data, makes it private and provides access methods. Any other "business" logic would go here as well

```swift
struct LegendsOfTomorrow {
    private var actors:[Actor] = [Actor(role: "Black Canary", name: "Caity Lotz"),
                                  Actor(role: "Captain Cold", name: "Wentworth Miller"),
                                  Actor(role: "Hawkgirl", name: "Ciara Renée"),
                                  Actor(role: "Heat Wave", name: "Dominic Purcell")]
    private var writers:[Writer] = [Writer(name: "Phil Klemmer", yearsWriting: 10),
                                    Writer(name:"Sarah Nicole Jones", yearsWriting:15),
                                    Writer(name:"Ray Utarnachitt", yearsWriting:12)]

    let yearsToBecomeExpert:Int = 10

    func numActors()->Int {
        return actors.count
    }

    func numWriters()->Int {
        return writers.count
    }

    func actor(_ index:Int)->Actor{
        return actors[index]
    }

    func writer(_ index:Int)->Writer{
        return writers[index]
    }

    func goodWriter(writer:Writer)->Bool {
        return writer.yearsWriting > yearsToBecomeExpert
    }

    mutating func addActor(_ actor:Actor){
        actors.append(actor)
    }

    mutating func addWriter(_ writer:Writer){
        writers.append(writer)
    }
}
```

```swift
struct Actor {
    let role:String
    let name:String
}

struct Writer {
    let name:String
    let yearsWriting:Int
}
```

LegendsOfTomorrow.swift

Creating the Model

- We can create the model in several places:

  1. Inside the TableViewController
  2. Inside the AppDelegate
  3. Inside itself
  4. Inside itself, using a singleton

Creating the Model: Inside the TableViewController

```
class LegendsTableViewController: UITableViewController {

    var legends:LegendsOfTomorrow = LegendsOfTomorrow()

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

- The View Controller, which *already* has a lot to do, is now tasked with creating and housing the model, too. No rest for the weary ...
- Q: What if other classes want to access this model? We could declare it as static, but that ties the model to this specific class (LegendTableViewController.legends) which contradicts the notion of separation of responsibilities

Creating the Model: Inside the AppDelegate

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    static var legends:LegendsOfTomorrow = LegendsOfTomorrow()


    func application(_ application: UIApplication,
          didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?)
                               -> Bool {
       return true
    }
```

```
class LegendsTableViewController: UITableViewController {



    override func numberOfSections(in tableView: UITableView) -> Int {
        return 2
    }

    // returns # of rows for each section (we have 1 section, with legends.count rows)
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if section == 0 {
            return AppDelegate.legends.numActors() // this is the total number of cells
        } else {
            return AppDelegate.legends.numWriters() //
        }
    }
```

Creating the Model: Inside Itself

```
struct LegendsOfTomorrow {

    static var legends:LegendsOfTomorrow = LegendsOfTomorrow()

    private var actors:[Actor] = [Actor(role: "Black Canary", name: "Caity Lotz"),
                                  Actor(role: "Captain Cold", name: "Wentworth Miller"),
                                  Actor(role: "Hawkgirl", name: "Ciara Renée"),
                                  Actor(role: "Heat Wave", name: "Dominic Purcell")]
    private var writers:[Writer] = [Writer(name: "Phil Klemmer", yearsWriting: 10),
                                    Writer(name:"Sarah Nicole Jones", yearsWriting:15),
                                    Writer(name:"Ray Utarnachitt", yearsWriting:12)]
    // ... etc
```

```
class LegendsTableViewController: UITableViewController {


    override func numberOfSections(in tableView: UITableView) -> Int {
        return 2
    }

    // returns # of rows for each section (we have 1 section, with legends.count rows)
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if section == 0 {
            return LegendsOfTomorrow.legends.numActors() // this is the total number of cells
        } else {
            return LegendsOfTomorrow.legends.numWriters() //
        }
    }
```

Creating the Model: Inside Itself, Using a Singleton

- The previous model, while it contained a static instance, does not stop us from creating another one: so in theory you could have one app with multiple versions of the model 🙁
- The singleton design pattern prevents that 🙂

Creating the Model: Inside Itself, Using a Singleton

```
struct LegendsOfTomorrow {

    private init(){} // stops anyone outside the struct from making a LegendsOfTomorrow instance
    static var legends:LegendsOfTomorrow = LegendsOfTomorrow()

    private var actors:[Actor] = [Actor(role: "Black Canary", name: "Caity Lotz"),
                                  Actor(role: "Captain Cold", name: "Wentworth Miller"),
                                  Actor(role: "Hawkgirl", name: "Ciara Renée"),
                                  Actor(role: "Heat Wave", name: "Dominic Purcell")]
// ... etc
```

- Q: Want to instantiate your own LegendsOfTomorrow?
- A: It can't happen, as init() is private. You *must* use the (public) static instance.

```
class LegendsTableViewController: UITableViewController {

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 2
    }

    // returns # of rows for each section (we have 1 section, with legends.count rows)
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if section == 0 {
            return LegendsOfTomorrow.legends.numActors() // this is the total number of cells
        } else {
            return LegendsOfTomorrow.legends.numWriters() //
        }
    }
```

Techy Aside: Initializers and Structs

All struct properties must have values before it is initialized

```
struct Size {
    private var width:Double
    private var height:Double
}
let size0 = Size() // Cannot invoke initializer for type 'Size' with no arguments
let size:Size = Size(width:0.0,height:0.0) // 'Size' initializer inaccessible due to 'private'
protection level
```

```
struct Circle {
    private var radius:Double = 0.0. // If members are initialized, however,
    private var filled:Bool = false  // we can still make a Circle :-(

}
let circle0 = Circle() // this works, but for singleton purposes we'd like to shut this down
```

```
struct Circle {
    private var radius:Double = 0.0
    private var filled:Bool = true
    private init(){}
}

let circle0 = Circle() // Ha! 'Circle' initializer inaccessible due to 'private' protection level
let circle1 = Circle(radius:0.0,filled:0.0) // cannot invoke initializer for type 'Circle' with
an argument list of type '(radius: Double, filled: Double)'
```

This slide clarifies why we need that private initializer on the previous slide. Some authors dislike singletons: see the Resources slide...

Multiple Sections

- It is possible to have multiple sections, each with its own header and footer. To accomplish this, implement these data source methods:

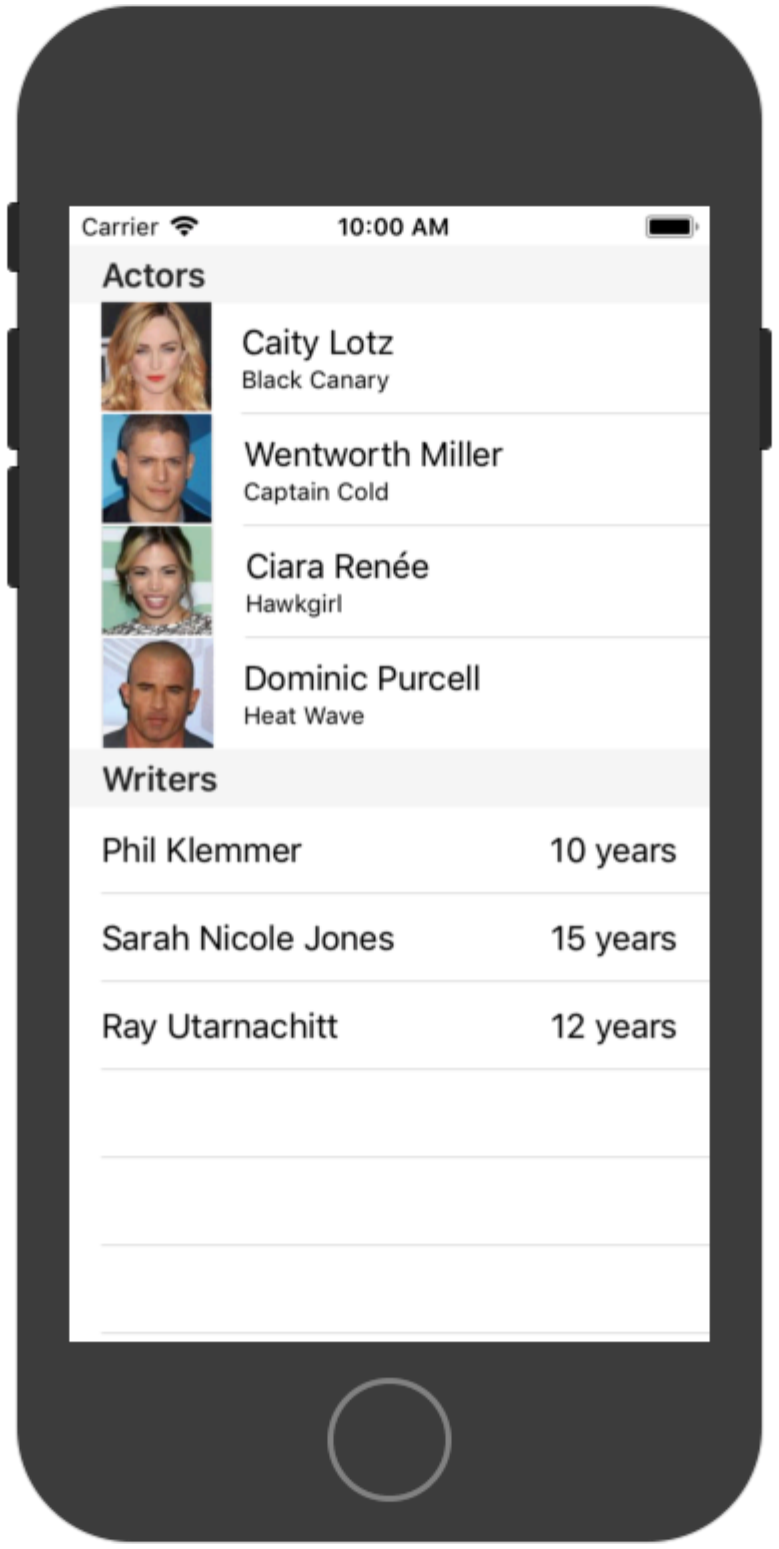
```
func numberOfSections(in: UITableView) -> Int
```
Returns the number of sections in the table view. If not specified, it defaults to 1

```
func tableView(UITableView, titleForHeaderInSection: Int) -> String?
```
Returns the title of the header of the specified section of the table view.

```
func tableView(UITableView, titleForFooterInSection: Int) -> String?
```
Returns the title of the footer of the specified section of the table view.

Multiple Sections: An Example

Plain Style



Carrier 10:00 AM

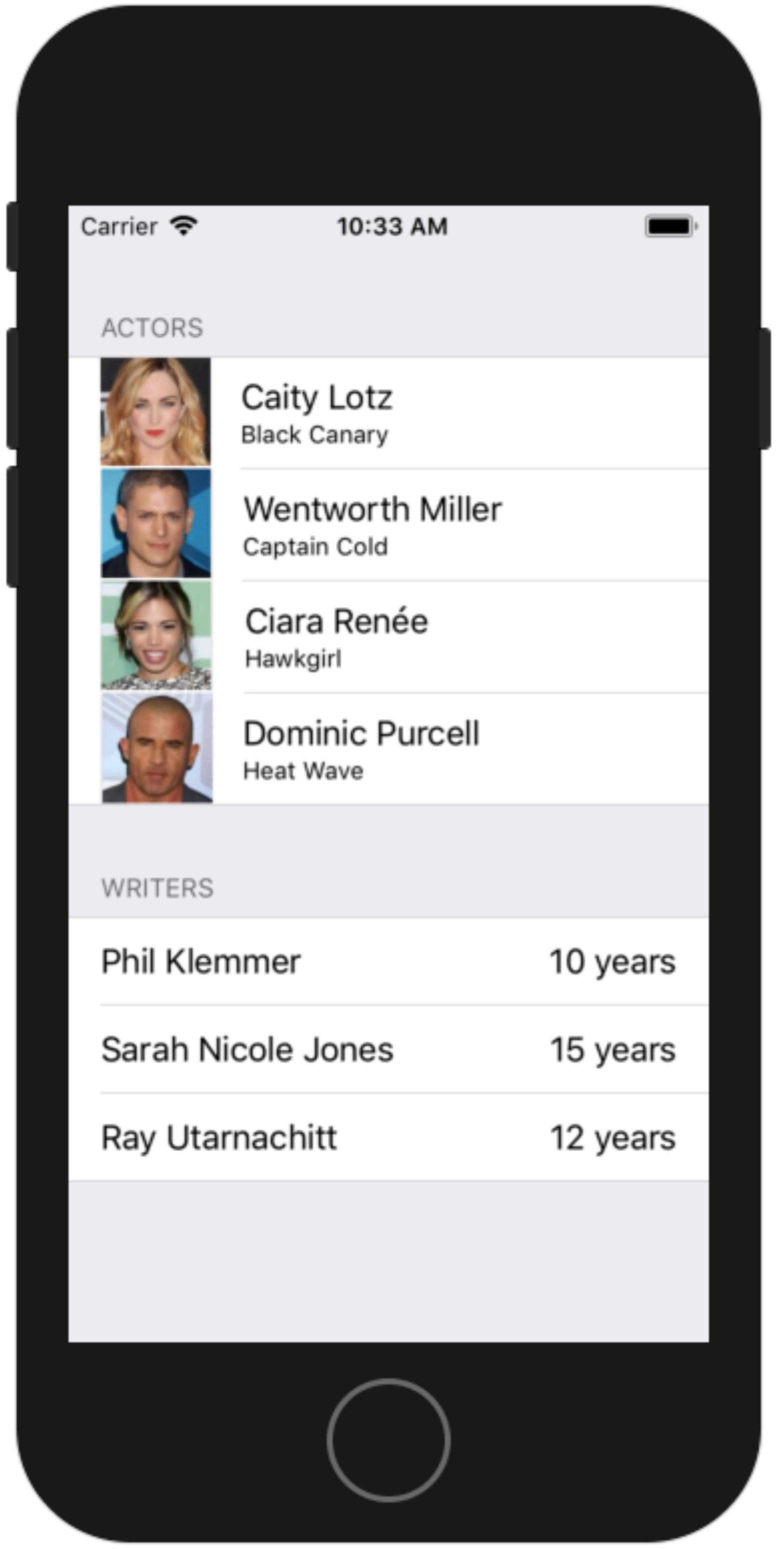
Actors

Caity Lotz	Black Canary
Wentworth Miller	Captain Cold
Ciara Renée	Hawkgirl
Dominic Purcell	Heat Wave

Writers

Phil Klemmer	10 years
Sarah Nicole Jones	15 years
Ray Utarnachitt	12 years

Grouped Style



Carrier 10:33 AM

ACTORS

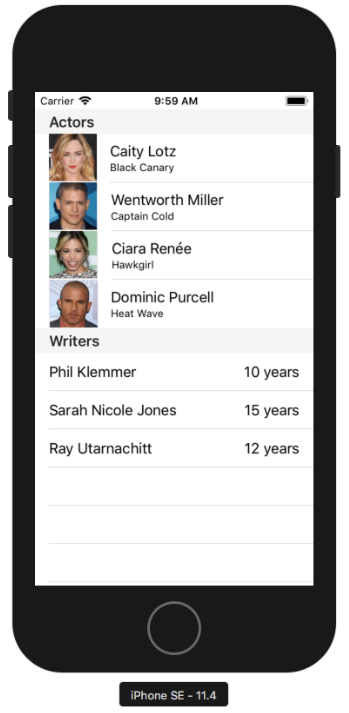
Caity Lotz	Black Canary
Wentworth Miller	Captain Cold
Ciara Renée	Hawkgirl
Dominic Purcell	Heat Wave

WRITERS

Phil Klemmer	10 years
Sarah Nicole Jones	15 years
Ray Utarnachitt	12 years

Multiple Sections: An Example

```
override func numberOfSections(in tableView: UITableView) -> Int {
        return 2
    }

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

        if section == 0 {
            return legends.numActors() // this is the total number of cells
        } else {
            return legends.numWriters() //
        }
    }

    override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int)
                                                                              -> String? {

        return ["Actors", "Writers"][section]
    }
```



Actors

Caity Lotz
Black Canary

Wentworth Miller
Captain Cold

Ciara Renée
Hawkgirl

Dominic Purcell
Heat Wave

Writers

Phil Klemmer	10 years
Sarah Nicole Jones	15 years
Ray Utarnachitt	12 years

Multiple Sections: An Example

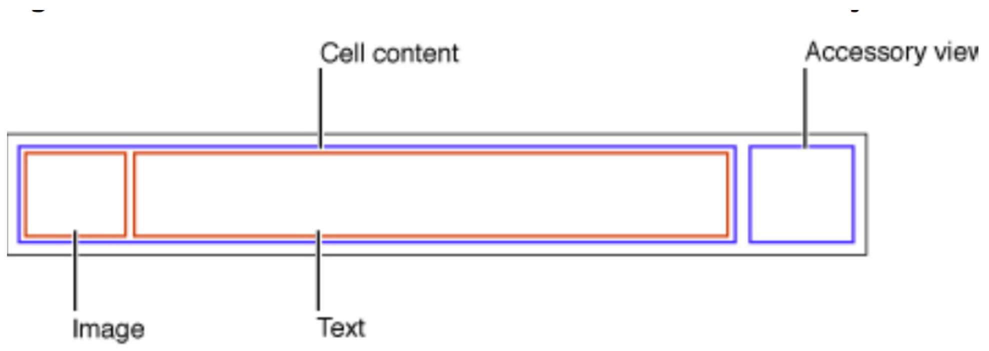
```swift
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
                                                                   ->  UITableViewCell {
if indexPath.section == 0 {
    let cell = tableView.dequeueReusableCell(withIdentifier: "actorCell")!
    cell.textLabel?.text = legends.actor(indexPath.row).name
    cell.detailTextLabel?.text = legends.actor(indexPath.row).role
    cell.imageView?.image = UIImage(named: "\(legends.actor(indexPath.row).name)")
    return cell
  } else {
    let cell = tableView.dequeueReusableCell(withIdentifier: "writerCell")!
    cell.textLabel?.text = legends.writer(indexPath.row).name
    cell.detailTextLabel?.text = "\(legends.writer(indexPath.row).yearsWriting) years"
    cell.imageView?.image = UIImage(named: "\(legends.writer(indexPath.row).name)")
    return cell
  }
}
```

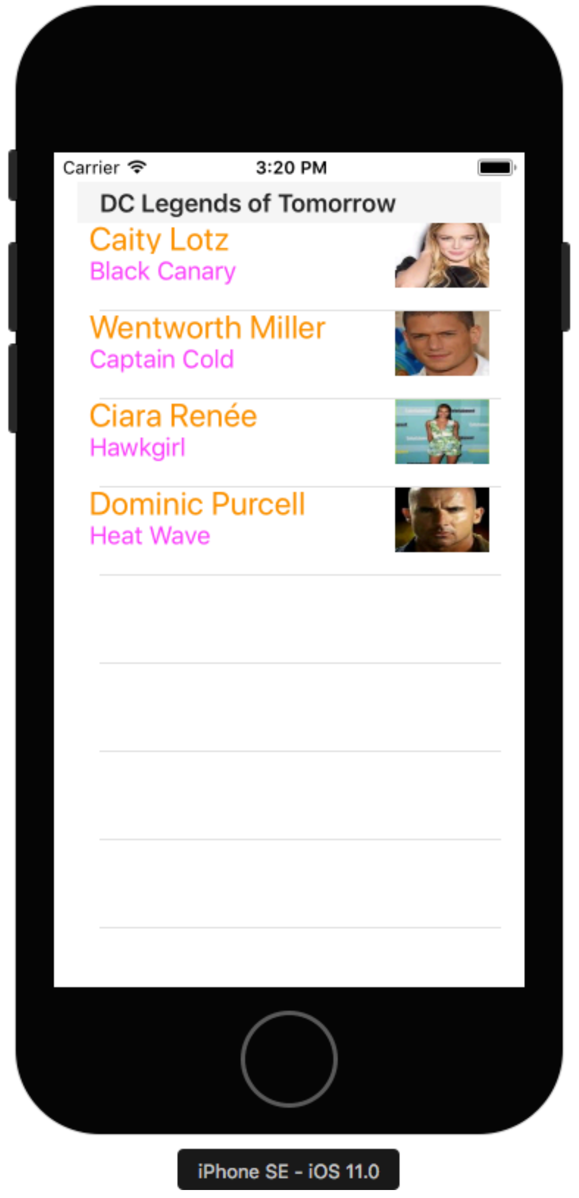
Custom Cell Types

- A UITableViewCell consists of a **contentView**, which depending upon the style can contain a textLabel, detailTextLabel, and imageView; and an **accessoryView** to display accessory images

- We can, in code or via storyboard, set the font, alignment, line-break mode and color of the UILabels

- But what if we want something *completely* different, for which no style exists? Say, an image view to the right of two vertically stacked labels? This is a job for ... Custom Cell Types (probably *not* a Legends superhero 😃)





DC Legends of Tomorrow

Caity Lotz
Black Canary

Wentworth Miller
Captain Cold

Ciara Renée
Hawkgirl

Dominic Purcell
Heat Wave

Custom Cell Types

1. Choose Custom for a prototype cell's style

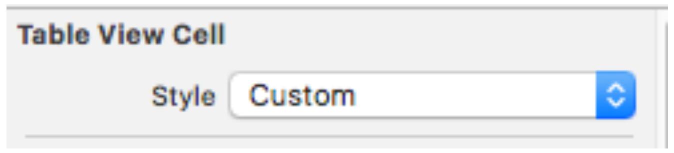
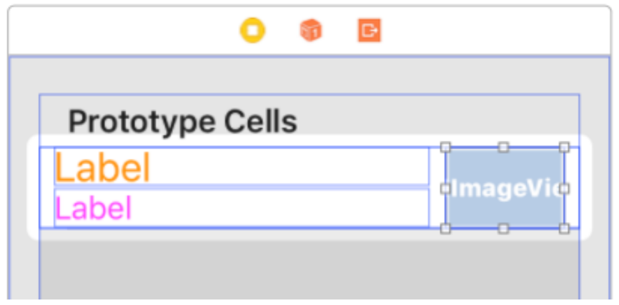


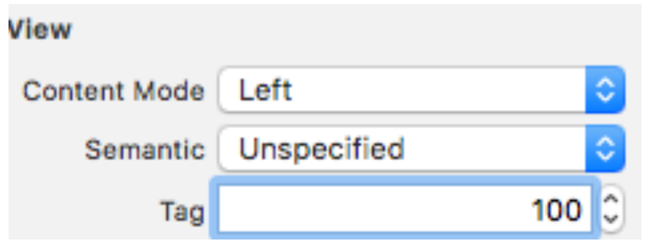
Table View Cell

Style: Custom

2. Drag any UI objects into the prototype cel's contentView and arrange them as needed



3. Assign each object a distinct **tag**, an integer to uniquely identify it



View

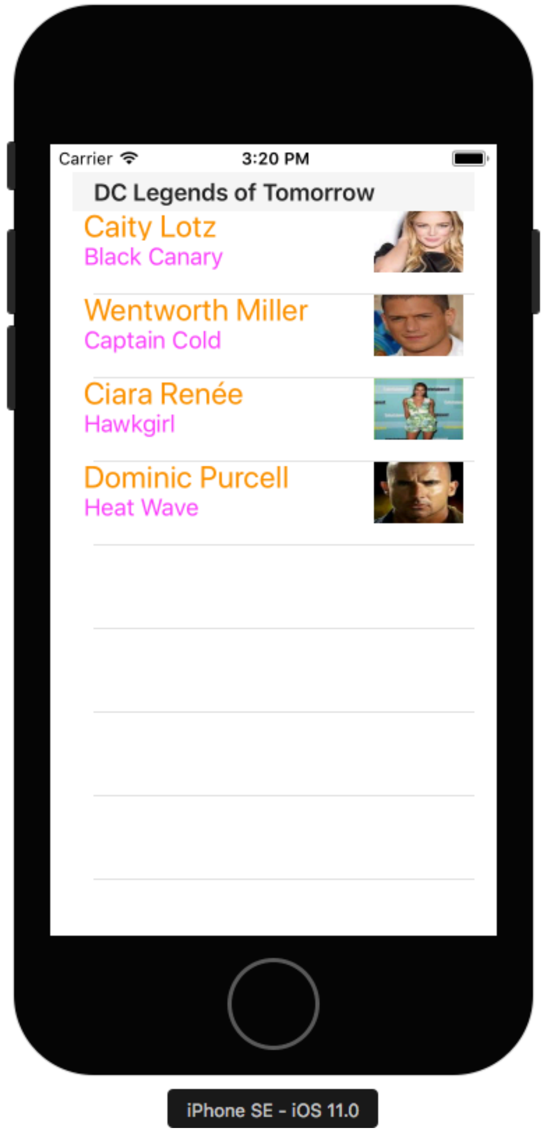
Content Mode: Left

Semantic: Unspecified

Tag: 100

4. Use those tags to retrieve them from the cell, and assign them values. See the next slide for an example

Custom Cell Types Example



DC Legends of Tomorrow

Caity Lotz
Black Canary

Wentworth Miller
Captain Cold

Ciara Renée
Hawkgirl

Dominic Purcell
Heat Wave

```
let desiredRowHeight = 60.0

func tableView(_ tableView: UITableView,
         heightForRowAt indexPath: IndexPath) -> CGFloat {
   return desiredRowHeight // another
    }

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

   let actorLBL = cell.viewWithTag(100) as! UILabel
   let actorIV = cell.viewWithTag(200) as! UIImageView
   let roleLBL = cell.viewWithTag(300) as! UILabel

   actorLBL.text = legends[indexPath.row].actor
   actorIV.image = UIImage(named: "\(legends[indexPath.row].actor).jpg")
   roleLBL.text = legends[indexPath.row].role

   return cell
    }
```

Resources

- developer.apple.com