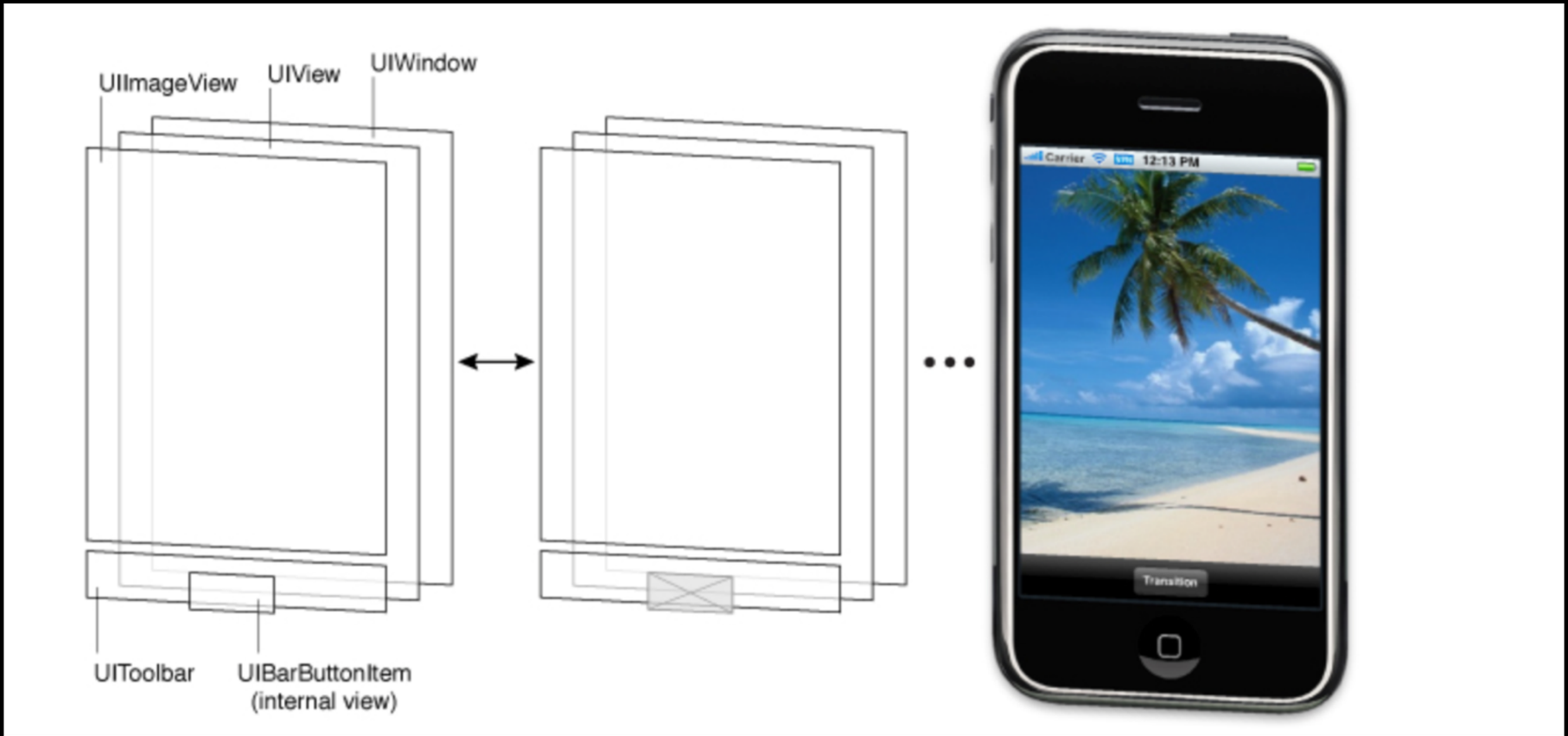
iOS Foundations II
Session 3

- View Hierarchy
- UITableView
- Delegation
- Arrays

Views



- “A view is an instance of the `UIView` class (or one of its subclasses) and manages a rectangular area in your application window”
- “Views are responsible for drawing content, handling multitouch events, and managing the layout of any subviews. “

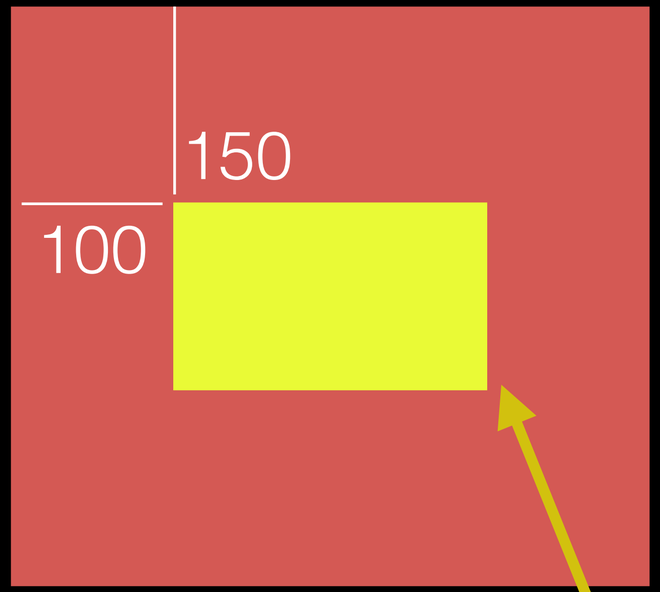
Views

- To get onscreen, a UIView must be added as a subview of a parent view. All views can be a child view of a parent view, and all views can have children views of their own. This is called the view hierarchy.
- A UIView has many properties that dictate its appearance: backgroundColor,alpha,hidden,opaque,tintColor,layer,clipsToBounds, etc.
- But the most important properties of a UIView have to do with location and size: Frame and Bounds

Frame and Bounds

- **Frame** - Describes the view's location and size **in its superview's coordinate system.**
- **Bounds** - Describes the view's location and size **in its own coordinate system.**
- Both of these properties are of type CGRect.
- A CGRect has 4 values: X, Y, Width, Height.
- Demo

Frame and Bounds

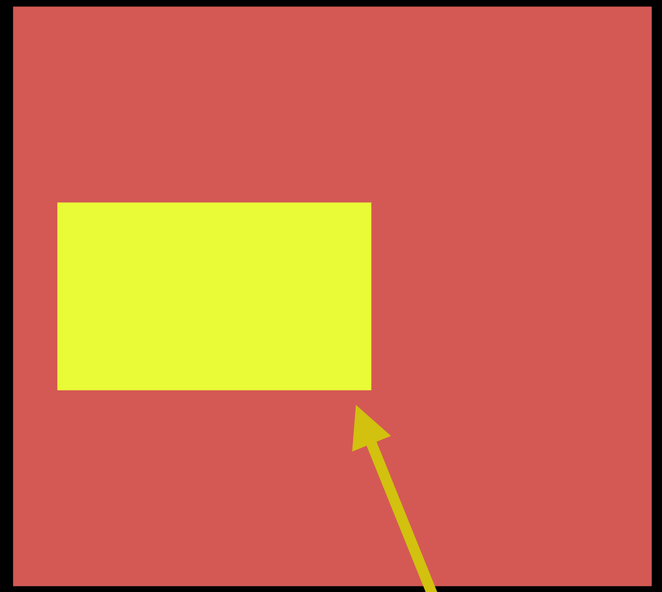


SuperView's Frame

- x = 0
- y = 0
- Width = 300
- Height = 300

ChildView's Frame

- x = 100
- y = 130
- Width = 150
- Height = 100



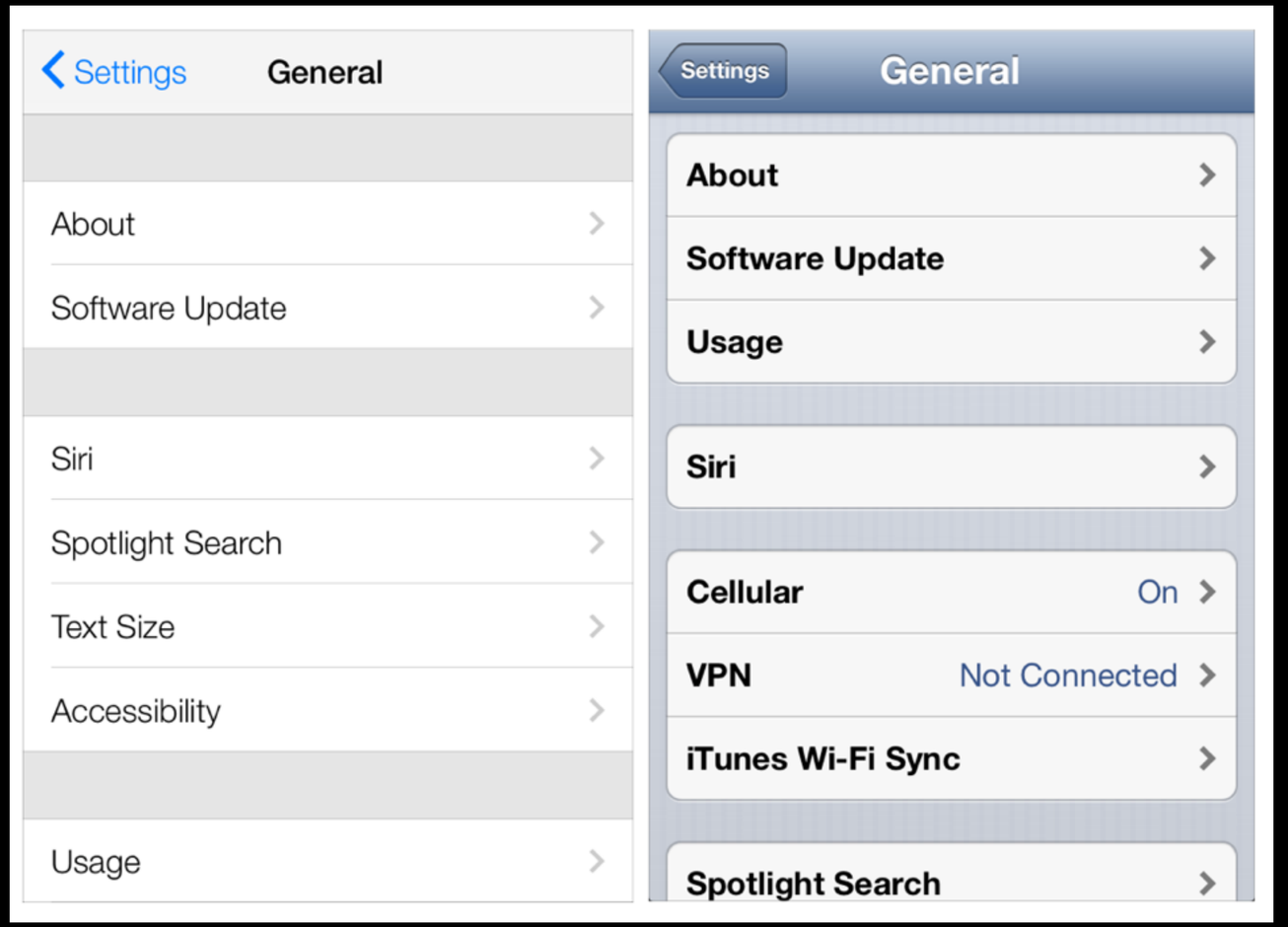
SuperView's Bounds

- x = 75
- y = 0
- Width = 300
- Height = 300

ChildView's Frame

- x = 100
- y = 130
- Width = 150
- Height = 100

UITableView and Delegation



TableViews

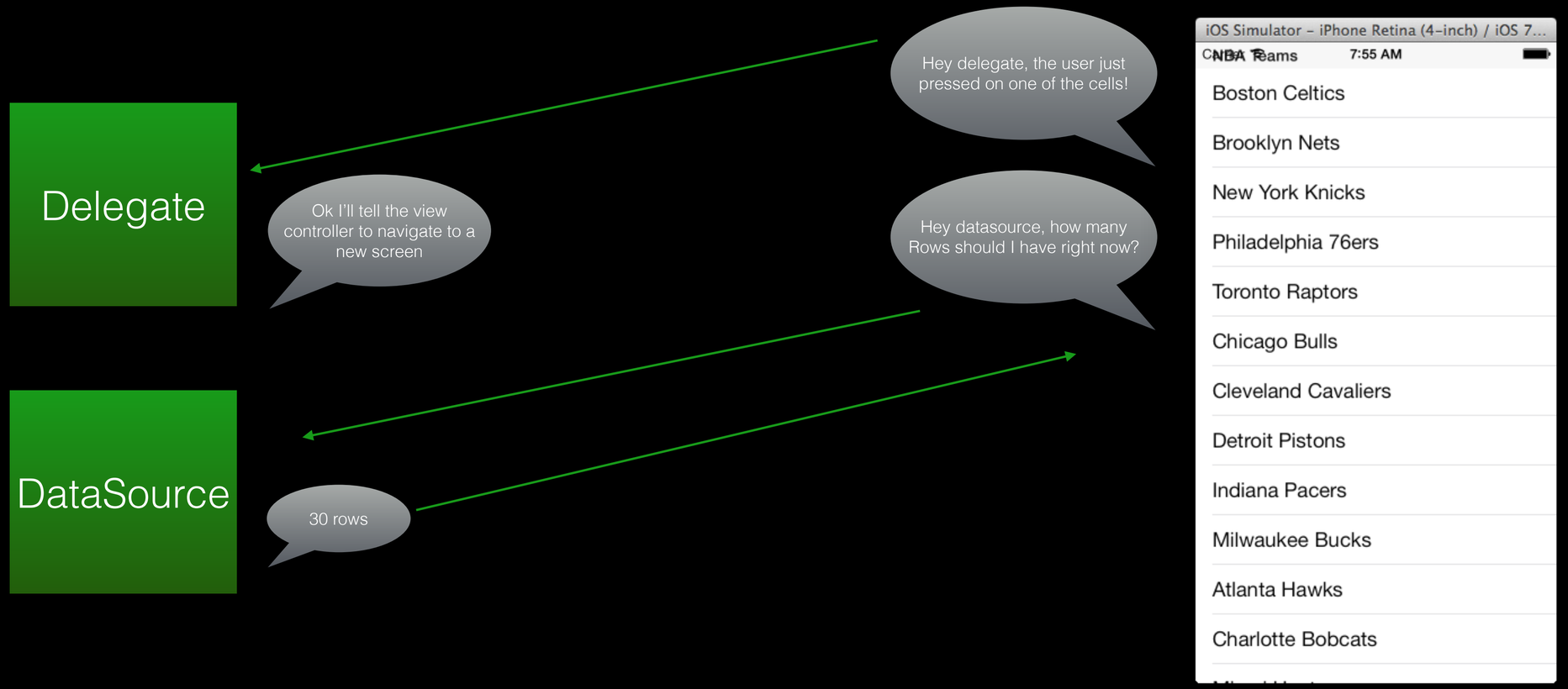
- “A Tableview presents data in a scrollable list of multiple rows that may be divided into sections.”
- A Tableview only has one column and only scrolls vertically.
- A Tableview has 0 through n-1 sections, and those sections have 0 through n-1 rows. A lot of the time you will just have 1 section and its corresponding rows.
- Sections are displayed with headers and footers, and rows are displayed with Tableview Cells, both of which are just a subclass of UIView.

So how do Tableviews work?

- Tableviews rely on the concept of delegation to get their job done.

- Picture time:

TableViews and Delegates



A tableview has 2 delegate objects. One actually called delegate and one called datasource. The datasource pattern is just a specialized form of delegation that is for data retrieval only.

Delegation

- The whole point of delegation is to allow you to implement custom behavior without having to subclass.
- Apple could have designed UITableView's api so you would have to subclass UITableView, but then you would have to understand UITableviews in a lot more detail(which methods can I override? Do I have to call super? omfg?)
- Delegates must adopt the protocol of the object they are being delegated from.
- Delegation is used extensively in a large portion of Apple's frameworks.

TableViews

- A tableview requires 2 questions to be answered (aka methods to be implemented) and they are all in the datasource. At the very least the tableview needs data to display. It doesn't have to respond to user interaction, so the delegate object is completely optional.
- `tableView(numberOfRowsInSection:) How many rows am I going to display?`
- `tableView(cellForRowAtIndexPath:) What cell do you want for the row at this index?`
- Number of sections is actually optional, and is 1 by default.
- Almost always, a tableview refers to array(s) to figure out what to display in each section and row. These are sometimes referred to as backing arrays.

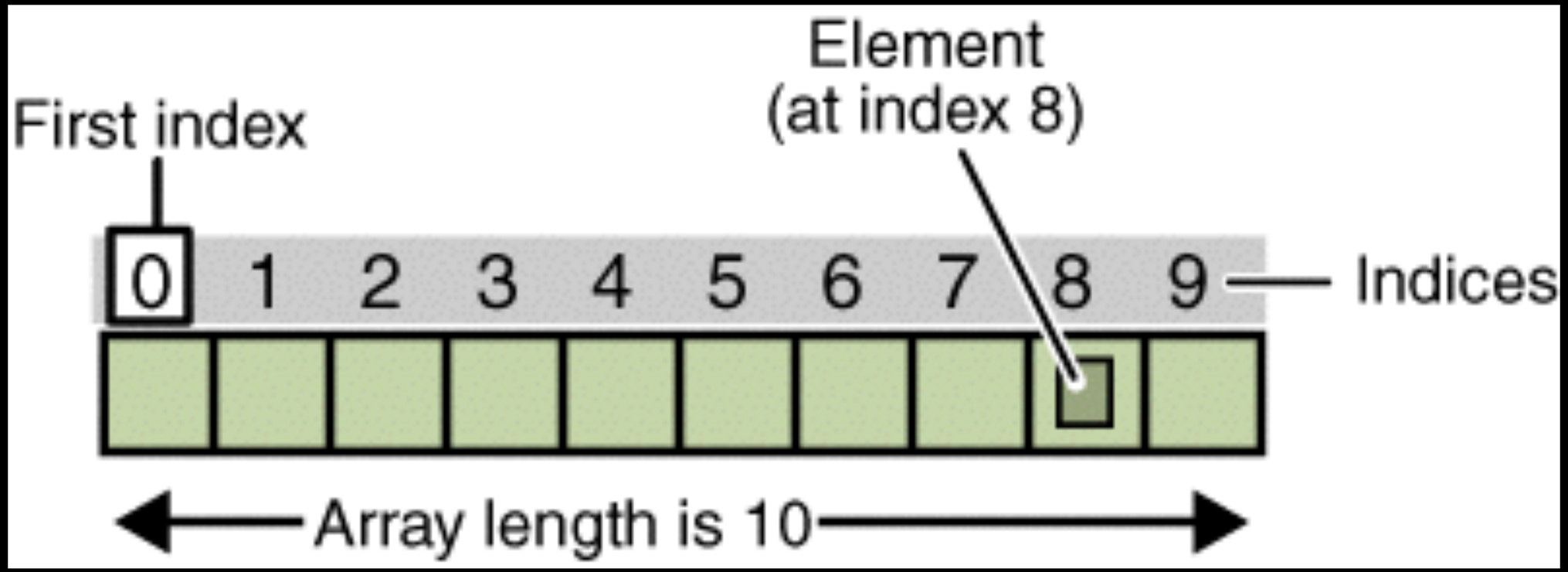
TableViews Gotchas

1. Did you give the tableview a data source? (by setting up an outlet to it and then settings it datasource property)
2. Did you conform to the data source protocol?
3. Did you implement the required 2 methods? (how many rows, and what cell for each row)
4. Did you drag out a tableview cell, and then set its reuse identifier?
5. Does the reuse identifier in your storyboard match the one in your code? “Did you forget your quotes in your code?” -Vick

Arrays

- Arrays are very important to understand
- Arrays are used in virtually ever app ever made!!!!
- You typically use arrays any time you have a collection of similar objects or data and you want to perform similar operations on them.
- So its considered a collection type. And its an object, **so it is passed by reference, not by copy.**
- Arrays are ordered, which is important.

Arrays


Element
(at index 8)
First index
0
1
2
3
4
5
6
7
8
9
Indices
Array length is 10

TableViews and Arrays

- When a tableview asks its datasource what it should display in each cell, it does so by sending a method for each cell that is going to be on screen.
- Think about how this would work without an array:

```
func tableView(tableView: UITableView!, cellForRowAtIndexPath indexPath: NSIndexPath!) -> UITableViewCell! {
    if indexPath.row == 0 {
       // return "Brad"
        }
    else if indexPath.row == 1 {
       // return "John"
        }
    else if indexPath.row == 2 {
        //return "Russell"
        }
    else if indexPath.row == 3{
            //return "Sherman"
        }
    else if indexPath.row == 4 {
        //return "Pete"
        }
}
```

TableViews and Arrays

- now the achieving the exact same functionality with an array:

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

    // return self.names[indexPath.row]
}
```