GENERAL ASSEMBLY

MOBILE DEVELOPMENT
WHILE LOOPS, STRUCTS, UIVIEW

Kishin Manglani

AGENDA

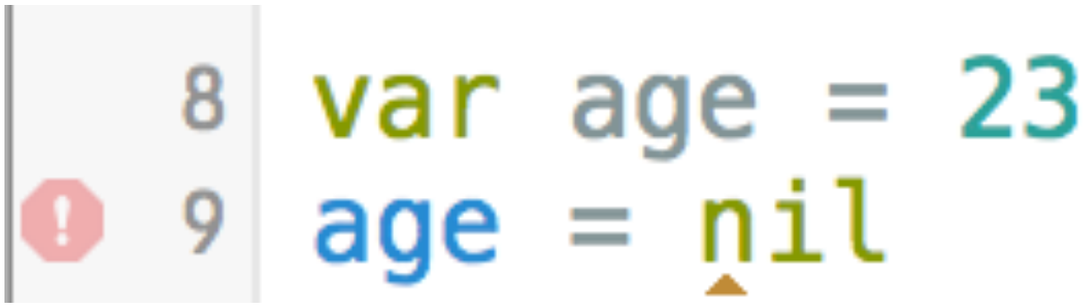
- Recap
- While Loops
- Computed Properties
- More on functions
- Structs
- CGGeometry
- Perform Segue With Identifier

RECAP

OPTIONALS

OPTIONALS

- So we actually need a name or object to represent nothing
- In Swift we use nil to represent nothing
- Before we could not assign nil to a variable



OPTIONALS

- Instead, to declare something as optional we can add a ? to the end of the type
- Adding the ? makes it an optional type, meaning that variable can now be assigned nil
- Remember, non-optional types are guaranteed to have an actual value

```
var height: Int? = 180
height = nil
```

```
var errorCode: Int?
```

OPTIONAL BINDING

- Here we can check to see if the optional is assigned a value:

```
if let constantName = someOptional {
    statements
}
```

FORCED UNWRAPPING

- Sometimes it's clear that our optional will ALWAYS have a valuable
- If we know that a value will always have a value, we can use an !
- We can just think of it as Swift saying, "Hey, I know you are an optional, and I know you have a value"
- This can cause errors if the value does not exist/is nil

```
var height: Int? = 180

func incrementInt(number: Int) -> Int {
    return number + 1
}

incrementInt(height!)
```

NIL COALESCING

- We can use a ?? to check to see if a value is nil
- If it is nil, we can assign it another value
- We can sort of think of this as a default value

```
var optionalInt: Int? = 33
var result = optionalInt ?? 0

optionalInt = nil
var result2 = optionalInt ?? 0
```

```
33
33

nil
0
```

OPTIONAL CHAINING

- If we use ! and the value doesn't exist our app can crash, that's why I call it a force unwrap
- When accessing properties we can use a ? instead of a !

```
myObject.employer?.companySize
```

OPTIONALS

- Why would we want to assign an object to nil?
- Sometimes things fail or sometimes things don't have values because it doesn't make sense
- What if had a division function? What if we divided by zero? Nil can be a good way to prevent that error

```
let possibleNumber = "123"

let convertedNumber = Int(possibleNumber)
```

OPTIONALS SUMMARY

- Nil represents the absence of a value
- In order to be assigned nil, they need to be of the Optional type. Non-optional variables and constants must have a non-nil value.
- To make something an Optional type we add a ? to the end of the type name
- Optional variables and constants are like boxes that can contain a value or be empty (nil)
- To use the value inside an optional, you must unwrap it from the optional
- If-let binding and nil coalescing are safer to unwrap optionals
- Forced unwrapping can produce a runtime error, so avoid when possible

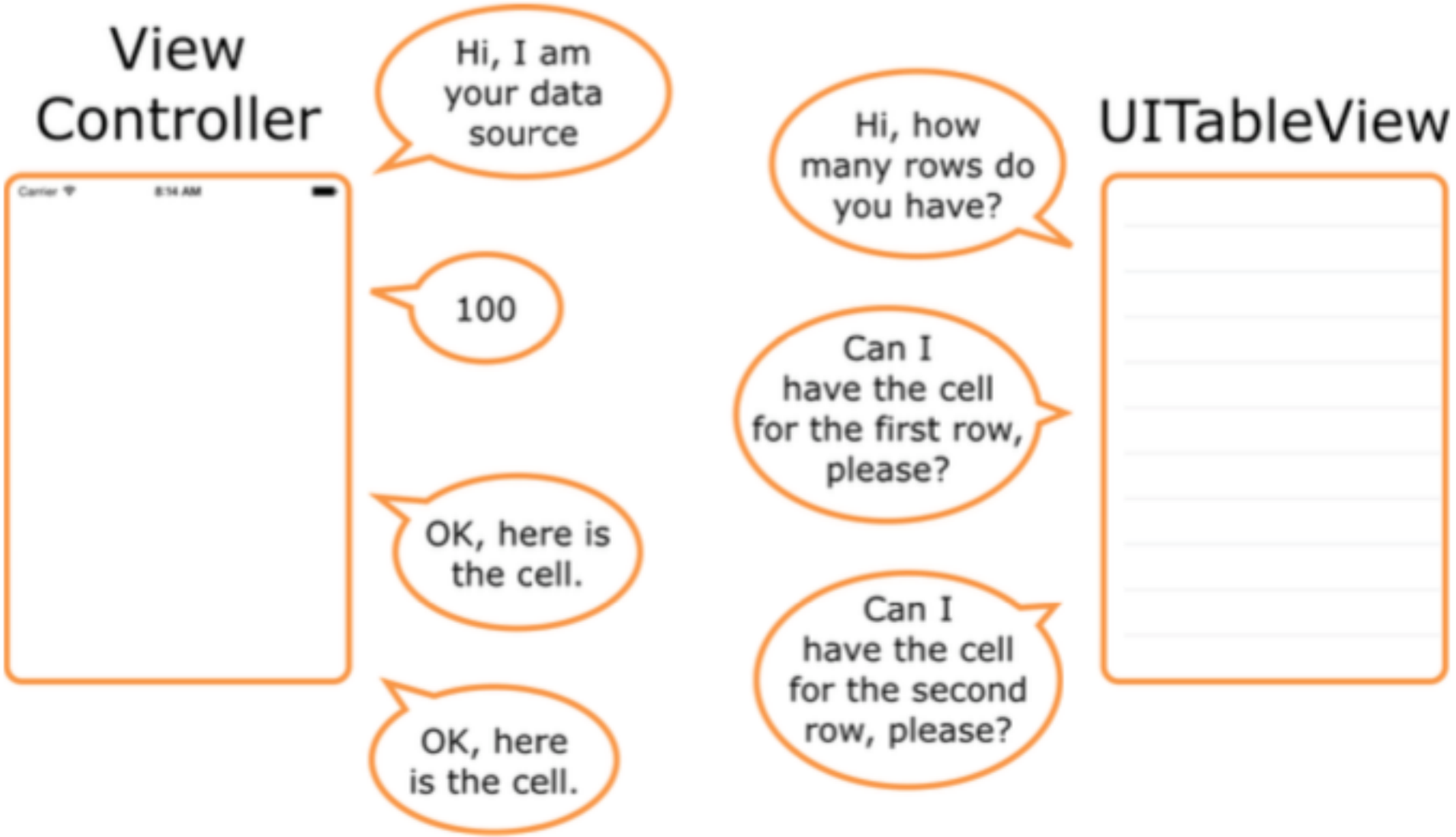
UITABLEVIEW

TABLE VIEWS

- Table views are a one dimensional list (list view may be a better name)
  - Vocabulary:
    - Section: All table views contain 1 or more sections; these are logical divisions of data
    - Row: Every section has a number of rows, which are entries in that section, each row has a UITableViewCell
    - Index path: The combination of a section and row that is a unique position in a table view
    - Cell: The view that is displayed for an index path (the class UITableViewCell is a subclass of UIView)
- Table views must have a number of sections, a number of cells in each section, and (optionally), the cells themselves
- Table views have a data source and a delegate (these are protocols)
  - Data source: Provides cells, number of cells and sections
  - Delegate: Gets called when things happen to the table view, provides some views (e.g. header and footer)

DATA SOURCE

- We are going to make our view controller adopt the UITableViewDataSource protocol
- That means our UIViewController subclass will NEED to implement a couple of methods (and can implement a few others optionally)
- Two required methods
  - tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  - tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int
- Cell for row at indexPath: returns a UITableViewCell
- Number of rows in section returns an Int telling the table view how many rows it will have


View Controller
Hi, I am your data source
100
OK, here is the cell.
OK, here is the cell.
Hi, how many rows do you have?
Can I have the cell for the first row, please?
Can I have the cell for the second row, please?
UITableView

WHILE LOOPS

WHILE LOOPS

- A while loop starts by evaluating a single condition. If the condition is true, a set of statements is repeated until the condition becomes false.

```
while condition {
    statements
}
```

REPEAT WHILE

- The other variation of the while loop, known as the repeat-while loop, performs a single pass through the loop block first, before considering the loop's condition. It then continues to repeat the loop until the condition is false.

```
repeat {
    statements
} while condition
```

COMPUTED PROPERTIES

COMPUTED PROPERTIES

- A computed property is essentially a function disguised as a property
- Computes or recalculates every single time the property is used
- A computed property MUST have an explicit type

```
class Square {
    var sideLength = 100

    var area: Int {
        get {
            return sideLength * 2
        }
    }
}
```

COMPUTED PROPERTIES

- We can use computed properties when both getting and setting a variable

```
class Square {
    var sideLength = 100

    var area: Int {
        get {
            return sideLength * 2
        }
        set(newArea) {
            sideLength = newArea / 2
        }
    }
}
```

COMPUTED PROPERTIES

- When is a good time to use this?
- Perhaps when one property is highly dependent on another, that way when we update one variable we don’t need to remember to update another

MORE ON FUNCTIONS

EXTERNAL VARIABLE NAMES

- When we added this to our UIViewController after adopting the UITableViewDataSource protocol:

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

  Some of you asked why there is cellForRowAtIndexPath and indexPath… it looks like there are two labels for the variable
- The cellForRowAtIndexPath is what's called the external variable name: this is what objects that call the function will pass in
- The indexPath is called the internal variable name: this is what we use within the function

EXTERNAL VARIABLE NAMES

- When we added this to our UIViewController after adopting the UITableViewDataSource protocol:

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

  Some of you asked why there is cellForRowAtIndexPath and indexPath… it looks like there are two labels for the variable
- The cellForRowAtIndexPath is what's called the external variable name: this is what objects that call the function will pass in
- The indexPath is called the internal variable name: this is what we use within the function

```
self.tableView(tableView, cellForRowAtIndexPath: myIndexPath)
```

EXTERNAL VARIABLE NAMES

- Then within the body of the function, when we want to refer to the cellForRowAtIndexPath argument, we can just use indexPath
- In this case, we use indexPath instead of cellForRowAtIndexPath because it's much shorted and easier to type
- However, we use cellForRowAtIndexPath when calling the function because it's more descriptive

OMITTING EXTERNAL PARAMETER NAMES

- We can also do the exact opposite. Instead of specifying a name or a more descriptive name, we can omit the names
- To do this we set the external parameter name to an underscore
- When omitting an external parameter name we don't need to add the labels

```
func someFunction(firstParameterName: Int, _ secondParameterName: Int) {
    // function body goes here
    // firstParameterName and secondParameterName refer to
    // the argument values for the first and second parameters
}
someFunction(1, 2)
```

```
func someFunction(firstParameter: Int, secondParameter: Int) {
    //body
}
someFunction(2, secondParameter: 3)
```

INTRO TO SWIFT

```
func exampleFunction(firstName firstParameter: Int, secondParameter: Int) {
    //body
}
exampleFunction(firstName: 4, secondParameter: 9)
```

INTRO TO SWIFT

```
func similarFunction(firstName firstParameter: Int, secondName secondParameter: Int) {
    //body
}
similarFunction(firstName: 9, secondName: 9)
```

```
func anotherFunction(firstParameter: Int, _ secondParameter: Int) {
    //body
}
anotherFunction(2, 3)
```

STRUCTS

STRUCTS

- Structs are kind of like lightweight classes
- For example, if our building class or our restaurant class had something like a location property and location were composed of latitude and longitude (both Doubles) we could make location a struct

```
struct Location {
    var latitude: Double
    var longitude: Double
}
```

STRUCTS

- Swift auto generates a simple initializer for structs
- We can still add our own initializers though
- By the end of the initializer, the struct must have initial values set in all of its stored properties, unless we use optionals, exactly like a class
- We can also add methods to a struct
- Really, the primary difference between declaring a class and a struct is the keyword: instead of class, we use struct

STRUCTS

```
class SomeClass {
    // class definition goes here
}
struct SomeStructure {
    // structure definition goes here
}
```

STRUCTS

- The main difference is that structs are value types and classes are reference types.
- Instances of value types are copied whenever they're assigned or used as a function argument. Numbers, strings, arrays, dictionaries, enums, tuples, and structs are value types.
- Classes point to a reference
- Example

CGGEOMETRY

CGGEOMETRY

- UIView's make up everything we see on the screen
- Each UIView needs for numbers to draw itself: an x coordinate, a y coordinate, a width and a height
- We use CGGeometry: Core Graphics Geometry
- The primary data type in CGGeometry is CGFloat, which is really just a double

CGGEOMETRY

- CGPoint is a struct that represents a point in a two-dimensional coordinate system, typically an x coordinate and a y coordinate
- CGSize is a struct that represents the dimensions of width and height
- CGRect is a struct with both a CGPoint (origin) and a CGSize (size), representing a rectangle drawn from its origin point with the width and height of its size.
- To create a CGRect we pass in x, y, width and height values

CGGEOMETRY

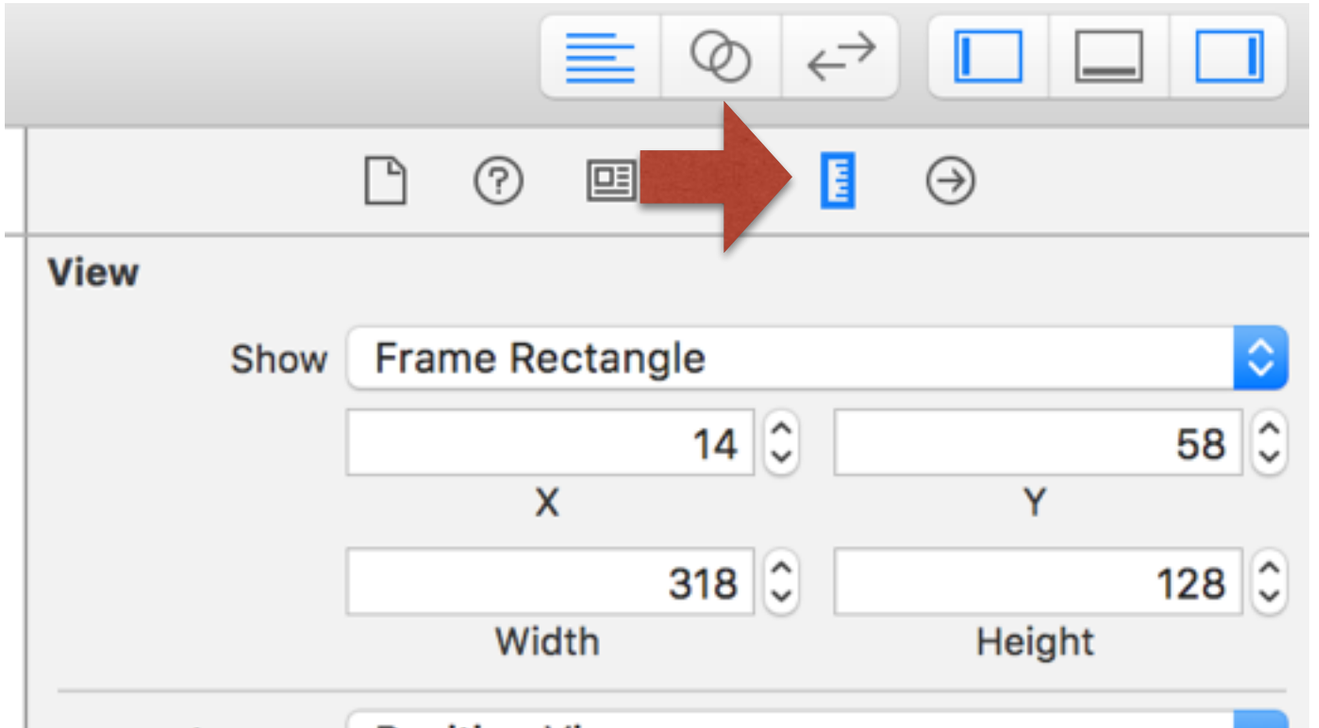
- UIView needs x, y, width and height as well
- To make this easier to deal with, UIView has a frame property and a bounds property
- Each of these is a CGRect, which again is composed of CGPoint and CGSize
- A view's frame (CGRect) is the position of its rectangle in the superview's coordinate system. By default it starts at the top left
- A view's bounds (CGRect) expresses a view rectangle in its own coordinate system.
- View's frame determines its location in superview. View's bounds determines its subviews locations. That means, if you change view's bounds, its location won't be changed, but all of its subviews location will be changed.

CGGEOMETRY

- We use the function CGRectMake to make a CGRect
- `let rect = CGRectMake(30, 25, 100, 100)`
- `let view = UIView(frame: rect)`
- `view.frame = rect`
- `view.backgroundColor = UIColor.blueColor()`

CGGEOMETRY

- Where do we set these values in Storyboard?

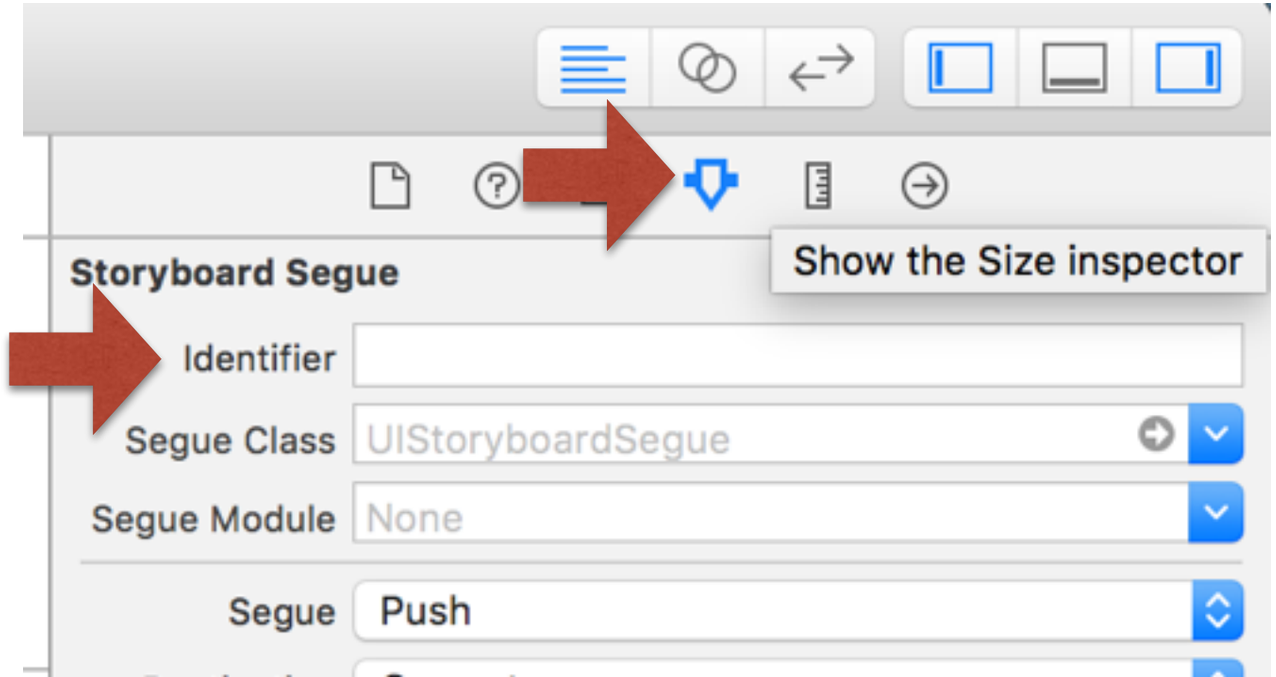


SEGUE

SEGUES

- We can set a specific identifier/name for a segue
- UIViewController has a method called:

```
self.performSegueWithIdentifier("identifier", sender: self)
```



PLAYGROUND

CHALLENGE

CHALLENGE

- Start a new project and add a UITableView
- Adopt the data source protocol: UITableViewDataSource
- Add the two required methods
- Instead of using the “Default” cell style try using the “Subtitle” style
- Also add text to the detailTextLabel of “Subtitle” style, what happens if you change back to Default?
- Add a second section to the UITableView
- In Storyboard change the table view’s “Style” from “Plain” to “Grouped”, what’s the difference?
- Change the row height of the UITableView in Storyboard
- Change the “Separator” property of the UITableView in Storyboard what are the differences
- Implement the UITableViewDelegate protocol
- Add the didSelectRowAtIndexPath method
- When a row is selected call the performSegueWithIdentifier method