GENERAL ASSEMBLY

MOBILE DEVELOPMENT
CONNECTING CODE TO INTERFACE BUILDER & INTRO TO CLASSES

Tedi Konda
Executive Director, Technology, Unison

LEARNING OBJECTIVES

- Review functions
- Create hooks from interface builder to Swift code
- Create and implement custom classes
- Access Xcode documentation for any external classes

REVIEW LESSON *5*

WHAT IS A FUNCTION?

- A function is a series of repeatable steps that, at some point, ends
- Optional input and output
- Multiple inputs and outputs, as needed

DEFINING FUNCTIONS

- func ***name***() { /* code */} // No parameters, no return
- func ***name***(***parameterName: type***) { /* code */} // One parameter, no return
- func ***name***(***parameterName: type, parameterTwoName: type***) { /* code */} // Two parameters, no return
- func ***name***(***parameterName: type***) -> ***returnType*** { /* code */} // One parameter, one returned value
- func ***name***() -> (***returnOne: valueOne***, ***returnTwo: valueTwo***) {/* code */} // No parameters, two returned values

CALLING FUNCTIONS

- ***name***() // No parameters, no return
- ***name***(***parameter***) // One parameter, no return
- ***name***(***parameter, parameterTwoName: parameterTwo***) // Two parameters, no return
- var result = ***name***(***parameter***) // One parameter, one returned value
- let result = ***name***() {/* code */} // No parameters, two returned values
- println("\(result.paramOneName) \(result.paramTwoName)")

FUNCTIONS RECAP

- Be descriptive: Name your functions with descriptive names and descriptive parameters
- Be brief: Keep your functions short (i.e. approximately less than a screen's worth of content). You should be able to describe what they do in once sentence
- Compose: Your functions can call each other
- DRY: Don't repeat yourself. Any time you find the urge to copy and paste, there may be an opportunity to break into a function

WHEN TO USE FUNCTIONS

- Functions are VERY common building blocks when writing code
- But figuring out how to break them up is HARD, even for intermediate developers
- Any time you find the urge to copy and paste
- Any time you have multiple parts of your application sharing the same functionality, or very similar functionality with different parameters
- KISS: Avoid the urge to over-compose. Over-composed code can be just as difficult to read as under-composed code

Group exercise: functions

HOOKING UP INTERFACE BUILDER TO CODE

STORYBOARDS

- Remember storyboards?
- Our view controllers in storyboards can be (and usually are) represented in code
- Our code can modify those view controllers, change its views, the properties of those views, etc
- We create the connections between our view controllers using ‘outlets’

IB'S FUNCTIONS

- Actions dragged into code from IB define **functions** (of a sort)
- Outlets dragged into code from IB define **variables** (of a sort)
- Your functions can interact with your variables
- Do you have a text field, label or text view that you've made an outlet of?
  - You can get its text by using 'var text = label.text'
  - You can set its text by using 'label.text = 'Some text' ‣ You can get and set other things, too!

XCODE DEMO: STORYBOARDS

DOCUMENTATION DEMO

CLASSES

WHAT IS THIS ‘CLASS’ KEYWORD?

- A basic building block
- A bundle of state and behavior that form an outline of a type
  - Variables (state)
  - Functions (behavior, in this case known as ‘methods’)
- One can create instances of classes
  - ‘Human’ is a class, ‘Tedi’ is the instance
- How does this tie into Interface builder and our code?

HOW DOES THIS TIE INTO IB?

- We use IB to set up various **classes** of controllers, and the **segues** that they use to connect with each other

- Uses these storyboards to create an **instance** of the first controller **class** when your app starts. That **instance** is what's displayed on screen.

- When you use a segue to go to a new view controller **class**, a new **instance** of it is created and navigated to

- Multiple **instances** of the same class can exist

XCODE DEMO: CREATING CLASSES