GENERAL ASSEMBLY

MOBILE DEVELOPMENT

INTRO TO FUNCTIONS

William Martin
Head of Product, Floored

Angel X. Moreno
EIR, Developer

LEARNING OBJECTIVES

- Review variables, types, values.
- Utilize “control flow” to make simple programs.
- Apply “Optionals” and explain when and how to use them.
- Identify functions and implement best practices
- Be able to call and define functions that take parameters
- Be able to use the returned value from a function
- Understand what returning from a function does

REVIEW LESSON 4

INTRO TO FUNCTIONS

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 are possible, as needed.
- Functions can contain variables visible only inside the function.
  - In general, a variable is only visible in the braces in which it was defined.

WHAT IS A FUNCTION?

- Let's say we want to run the same few lines of code in multiple situations. We might wrap those lines of code into a function, let's call ours *sayHello:*

```
func sayHello() {
    println("Hello!")
}
sayHello()
```

SYNTAX - SIMPLE FUNCTIONS

- Defining a function:

```
func sayHello() {
    /* our code here */
}
```

- Calling: `sayHello()`
- This runs all the code within the above function, then continues to the next line of code.

SYNTAX - FUNCTION WITH ONE PARAMETER

```
func sayHello(name:String) {
    println(“Hello! \(name)”)
}
sayHello(“Toshi”)
```

SYNTAX - FUNCTION WITH TWO PARAMETERS

```
func say(what:String, name:String) {
    println(“\(what)! \(name)”)
}
sayHello(“Hello”, name:“Toshi”)
```

SYNTAX - FUNCTIONS WITH PARAMETERS

- Let's say we want to run the same few lines of code in multiple situations, and a few variables in those lines of code vary across situations. We might wrap those lines of code into a function that takes some input (i.e. parameters)
- Defining: func *name*(*parameterName*: *Type*) { /* code */} // One parameter
- Defining: func *name*(*parameterName*: *Type*, *parameterTwoName*: *Type*) { /* code */} // Two parameters
- Calling: *name*(*parameter*) // One parameter
- Calling: *name*(*parameter*, *parameterTwoName*: *parameterTwo*) // Two parameters

SYNTAX - RETURN

- Let's say we're interested in the result of a certain bit of code, we might want that code to return a value which the calling code can capture and use
- Defining: func *name*() -> *ReturnType* { /* code */} // return ReturnType
- Defining: func *name*() -> (*ReturnType*, *ReturnType*) { /* code */} // returns a tuple or ReturnTypes
- Calling: var *value* = *name*() // Value is of type *ReturnType*

SYNTAX - 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*() // No parameters, two returned values
  - println("\(result.paramOneName) \(result.paramTwoName)")

SYNTAX – 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

XCODE DEMO: FUNCTIONS

GETTING STARTED

EXERCISE

KEY OBJECTIVE(S)

Create and use functions with parameters and return values.

TIMING

30 min	1. Code with partner
5 min	2. Debrief

DELIVERABLE

To the best of your ability, complete the provided playground file. If you hit a question you don't feel comfortable with, ask an instructor.

FUNCTIONS RECAP

- Functions are blocks of code that are runnable from anywhere where the function is visible
- When a function is called from within our code, code execution steps into the function until it returns
- Functions can take parameters and return values
- When defining a function, **return** stops all execution of the function and kicks back out to the caller

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.

GETTING STARTED

EXERCISE

KEY OBJECTIVE(S)

Create and use functions with parameters and return values.

TIMING

30 min	1. Code with partner
5 min	2. Debrief

DELIVERABLE

To the best of your ability, complete the provided playground file. If you hit a question you don't feel comfortable with, ask an instructor.