GENERAL ASSEMBLY

INTRO TO IOS PROGRAMMING WITH SWIFT

John Clem, CTO FiLMiC Pro

AGENDA

5:50 - 6:00	Download files from github.com/johnnyclem/intro-to-ios
6:00 - 6:45	Intro to Swift
6:45 - 6:55	BREAK
6:55 - 7:40	Intermediate Swift
7:45 - 8:15	Live Demos
8:15 - 9:00	Q&A, Team Challenge

SWIFT TIPS

- Apple wants your feedback
  (and reserves the right to make sudden changes to everything)
- Please ask relevant questions during the lecture
- Feel free to follow along in a Swift Playground text

MEET SWIFT

concise

expressive

safe

fast

modern

```
- (instancetype)init {
    if (self = [super init]) {
        self.firstName = @"Default First Name";
    }
    return self;
}
@property (nonatomic, strong) NSString *firstName;

var firstName = "Default First Name"
```

OBJECTIVE-C WITHOUT THE C

- Same runtime
- Same LLVM compiler & LLDB debugger
- Same Cocoa & Cocoa Touch Frameworks
- ‘Spiritual Successor’ to Objective-C

VARIABLES AND CONSTANTS

- var for variable variables (value can be set to a different value)
- let for constant variables (value cannot be set to different value)
- similar to mutable and immutable BUT
- let isn't just for constants

INT, UINT, FLOAT, DOUBLE, BOOL

- var pi : Float = 3.141592
- var precisePi : Double = 3.14159274101257
- var temperature : Int = -32
- var age : UInt = 31
- var alive : Bool = true

VAR: STRING

- var declares mutable String, let declares immutable
- copied when assigned (not a reference type like NSString *)
- String interpolation

```
var name = "johnny"                              “johnny"
name += "clem"                                   “johnnyclem"
var githubName = "@\(name)"                      “@johnnyclem"
var gmailName = "\(name)@gmail.com"              "johnnyclem@gmail.com"
```

VAR : STRING

- Strings are just unicode character arrays
- let cats = [😼, 😾, 😿]
- Iterating over a String

```
var name = "johnny"
for character in name {
    // do stuff with character
}
```

VAR : ARRAY<T>

- strongly typed array, not just a random-object-container
- use isEmpty property to check if count is 0
- use append() to add items to end of array
- += to combine two type-compatible arrays

VAR : DICTIONARY<T1, T2>

- Always specify the type of values AND keys
- Dictionary<KeyType, ValueType>
- .keys and .values properties (for enumerating)

FUNCTIONS

- self-contained chunks of code
- functions have names that are used to call the function
- parameters are comma separated

```
func doSomethingWith( object : AnyObject?) {
        object.doSomething()
}
```

FUNCTIONS

- parameters and return values are not required
- Tuples allow functions to return multiple values
- return type denoted with ->

```
func fullName( firstName : String, lastName : String) -> String {
        return firstName + “ “ + lastName
}
```

METHODS

- methods are functions associated with a type (class,type,enums)
- methods still use the func keyword!
- methods are called just like functions with one difference:
- parameter names in methods are also used when calling the method (except for the first one!!!)

TUPLES

- values in a tuple can be of any type and different types
- no limit on how many values inside
- use _ to ignore parts of the tuple when decomposing

TYPE INFERENCE

- if you don’t specify the type, Swift will work out the appropriate type.
- far fewer type declarations than Objective-C.
- give variables a default value, or declare their type
- AnyObject is like the Objective-C id or instancetype
- you can easily type-cast objects, as long as it’s a downcast

```
var tires = 4.0023
let tiresInt = tires as Int
```

BREAK