GENERAL ASSEMBLY

INTERMEDIATE SWIFT

John Clem, CTO FiLMiC Pro

CLOSURES { }

- before we can talk about closures, we need to talk about functions
- functions are first class citizens in Swift (you can have functions be parameters to another function, and you can return a function from another function)
- functions have a type, and the type is defined by the parameters + return type.
- String has a method toInt( ) that takes a String and returns an int. So that function's type is `(String) -> Int.`

CLOSURES { }

- functions are simply closures that don't do any value-capturing.
- nested functions capture values from their enclosing function
- closure expressions:

  like inline blocks are to Objective-C
  can capture values from their surrounding context

CLOSURE SYNTAX

flexible, but challenging (illegible) syntax

closure expression syntax:

{ (parameters) -> return type in statements

the in keyword is the start of the closure body

please refer to fuckingclosuresyntax.com

REFINING THE SYNTAX

Implicit Returns from Single-Expression Closures

if the closure only has one expression, you don't need the return

shorthand argument names.

$0, $1, $2, refer to the parameters in the closures body, and drop the argument list entirely. Thanks Dre.

TRAILING CLOSURES { }

You can write a trailing closure if

- the closure is the final argument of a function

Use trailing closure when you can't fit the closure on a single line

Outside & after the function parameters

```
networkController.fetchDataWithCompletion() {
        // completion stuff goes here
}
```

CAPTURING VALUES

- closures capture variables from the context they are defined in
- variables that are not modified are copied in (enclosed)
- variables that are modified are referenced and kept alive.

EXTENSIONS

- add new functionality to an existing class, structure, or enum.
- unlike categories in Objective-C, extensions do not have names
- useful for computed properties, methods, initializers, subscripts, new nested types
- use to make an existing type conform to a protocol

EXTENSIONS

- Computed properties don't actually store a value, just compute and return a value from existing data.
- Initializers: adds new convenience initializers, but not designated initializers.
- Methods: regular instance and type methods, or a mutating method that modifies self on structs or enums. Must be marked with mutating keyword.

EXTENSIONS (CONT)

Subscripts: add new subscripts to an existing class.

Nested Types. types, structs, and enums nested inside an extended type.

OPTIONALS?

- forced unwrapping (!) accesses the value inside an optional
- optional binding checks if it has a value before proceeding
- optional chaining validates the optional nested properties
  and aborts after first nil?
- Implicitly Unwrapped Optionals are used when you can be sure an optional! will have a value after initial setup
- downcasting with ‘as?’

OPTIONALS!?

- variables whose value may or may not be present
- Swift does not allow you to leave properties in an undetermined state.
- variables must either:
    - be given a default value
    - have their value set in the initializer
    - be marked as optional using either the ? or ! symbol

Q&A

THANKS!

JOHN CLEM

- johnnyclem@gmail.com
- @johnnyclem Twitter & Github
- learnswift.io