Introduction to Swift

Agenda

- **What is Swift and why should we use it?**
- Playgrounds
- Mutability and Optionals
- Functions
- Swift and Objective-C
- Wish List

What is Swift?

- For iOS and Mac app development
- Built on the Objective-C runtime and can interoperate with Objective-C code
- Multi-paradigm
- Compiled
- Static, strong, inferred type system
- ARC (Automatic Reference Counting) for memory management

Why Use Swift?

- Potentially faster than Objective-C
- More concise than Objective-C
- Helps you avoid the most common types of crashes that happen in Objective-C
- Because Apple says so

Playgrounds

- Are a REPL on steroids
- Support dynamic code evaluation
- But also support rich text and dynamic views

Mutability and Optionals

- `let` and `var` make mutability explicit
- Optionals help protect against dereferencing null

Functions

- Can have multiple return values (via support for tuples)
- Allow for different external and internal parameter names
- Are first class objects

Functional Programming

- Function types can be aliased
- Closures can be passed around (like blocks in Objective-C)
- Native collections support functional operations like map, filter, reduce, etc.
- `switch` allows pattern patching

Swift and Objective-C

- You can mix Swift and Objective-C code in the same project
- Xcode can help set up header files and project settings to facilitate interoperability
- Desk.com is successfully doing this in production today

Swift and Objective-C

- All Objective-C methods can implicitly accept or return `nil`, whereas Swift has explicit optionals
- Objective-C methods can accept or return objects of type `id`, which appears in Swift as `AnyObject?`
- Swift collection types will be automatically bridged to `NSObject` subclasses, but beware of collection types that can accept `nil` values (e.g., `[String?]`)
- Swift class must subclass `NSObject`
- `struct, enum` and other features are not available in Obj-C

Wish List

- Cleaner optional unwrapping
  - `if let foo { doSomethingWithUnwrappedFoo(foo) }`
  - `let foo = foo else { return 0 }`
- Pure Swift optional protocols
  - all optional protocols require the @objc prefix
- Better reflection
  - possible if you subclass `NSObject`, but not in pure Swift

Wish List

- Better tooling
  - Playgrounds are buggy
  - Compilation is slow
  - Debugging can be problematic
  - Code auto-completion is slow
  - Syntax highlighting is buggy