# Introduction to Swift

# [TalkingPoints]

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

# What is Swift?

## 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?

## Why Use Swift?

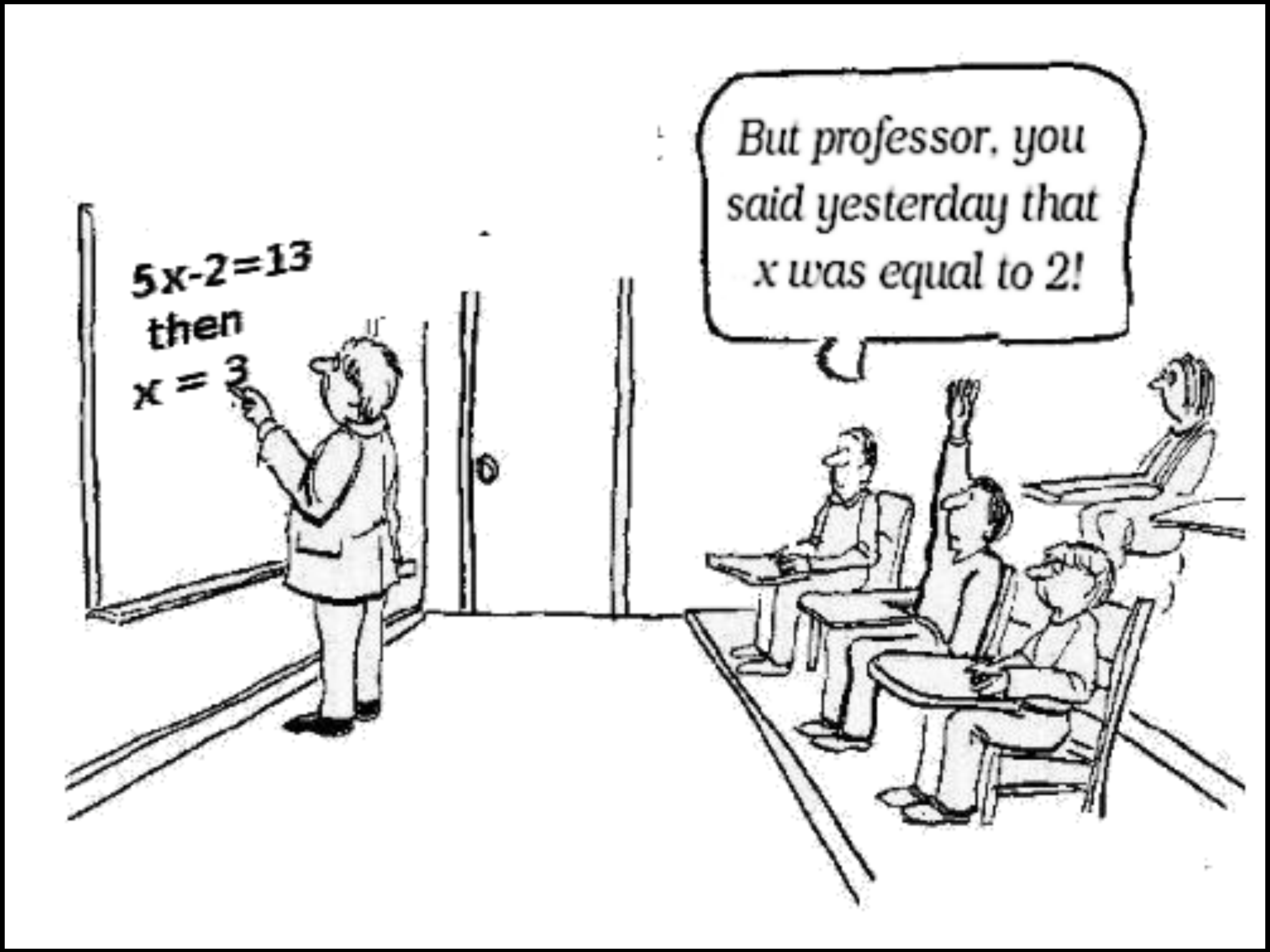
- Potentially faster than Objective-C
- More concise than Objective-C
- Helps you avoid the most common types of crashes that happen in Objective-C
- Because Apple says so

## Playgrounds

## Playgrounds

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

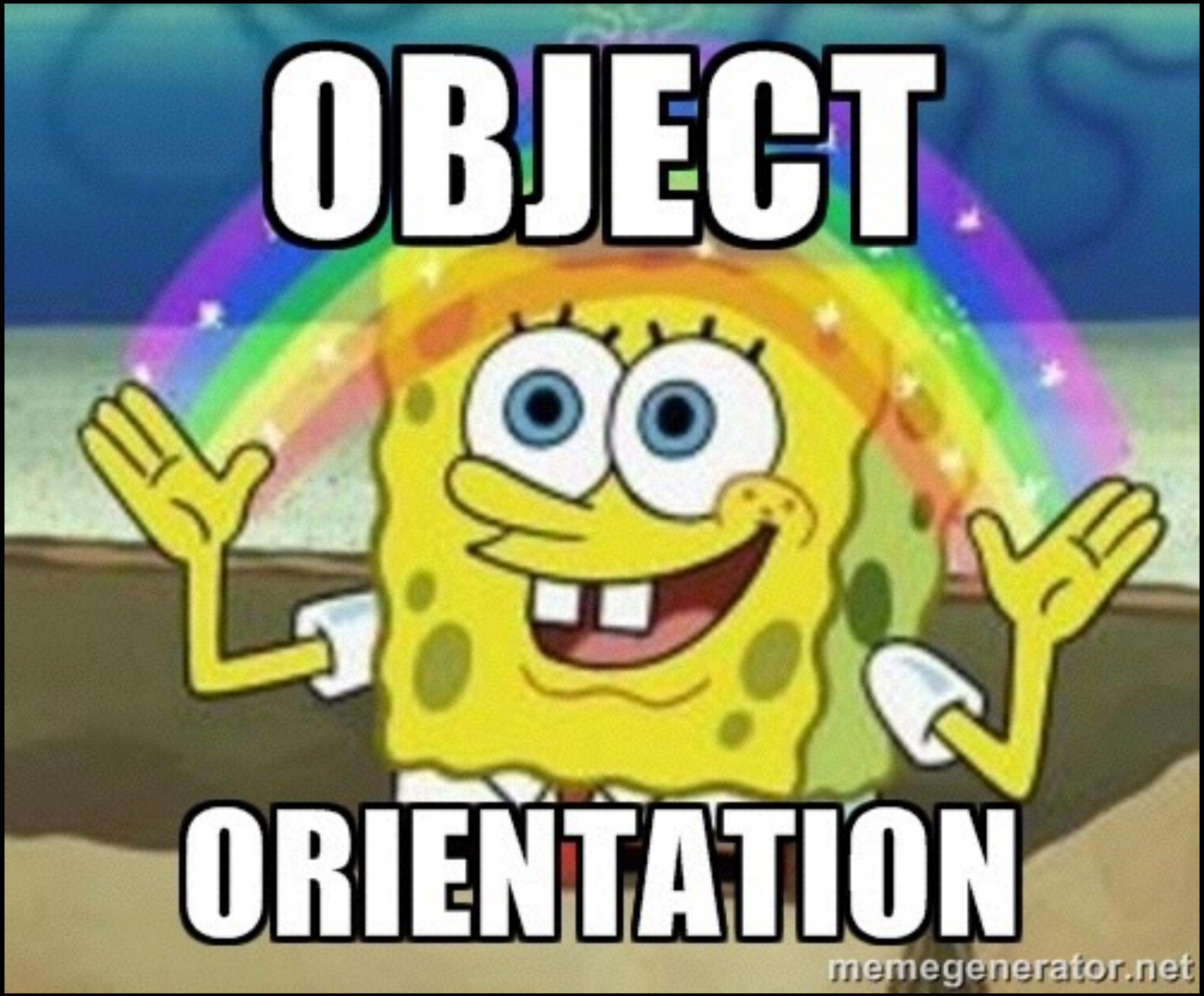
# Mutability and Optionals



## Mutability and Optionals

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

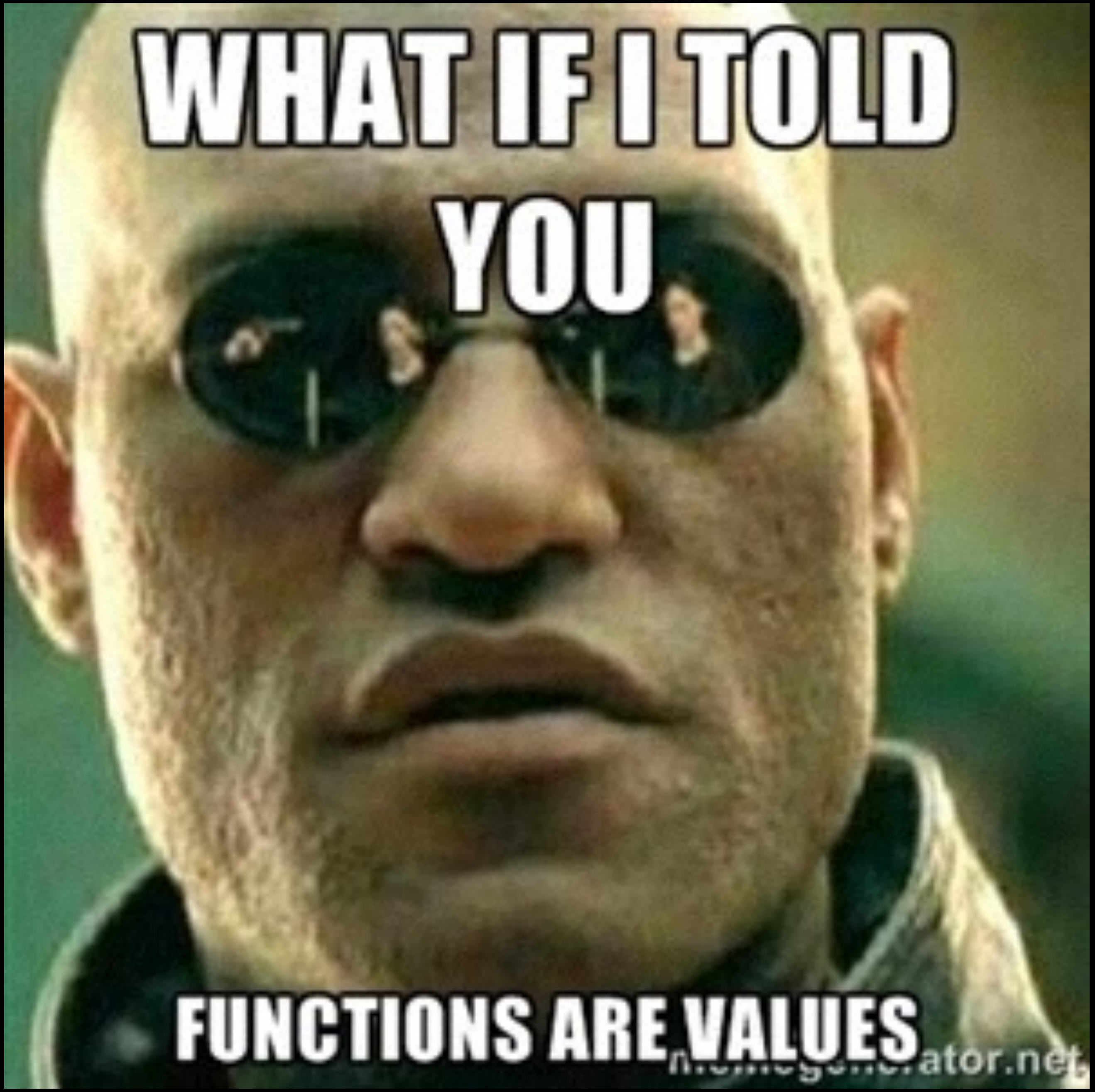
## Classes and Objects



# Classes and Objects

- `enum` is used to group a set of related values
- `struct` is used to represent a simple data structure
- `class` is used to represent a more complex data structure and object graph
- `enum` and `struct` are value types, while class is a reference type
- `class` provides inheritance

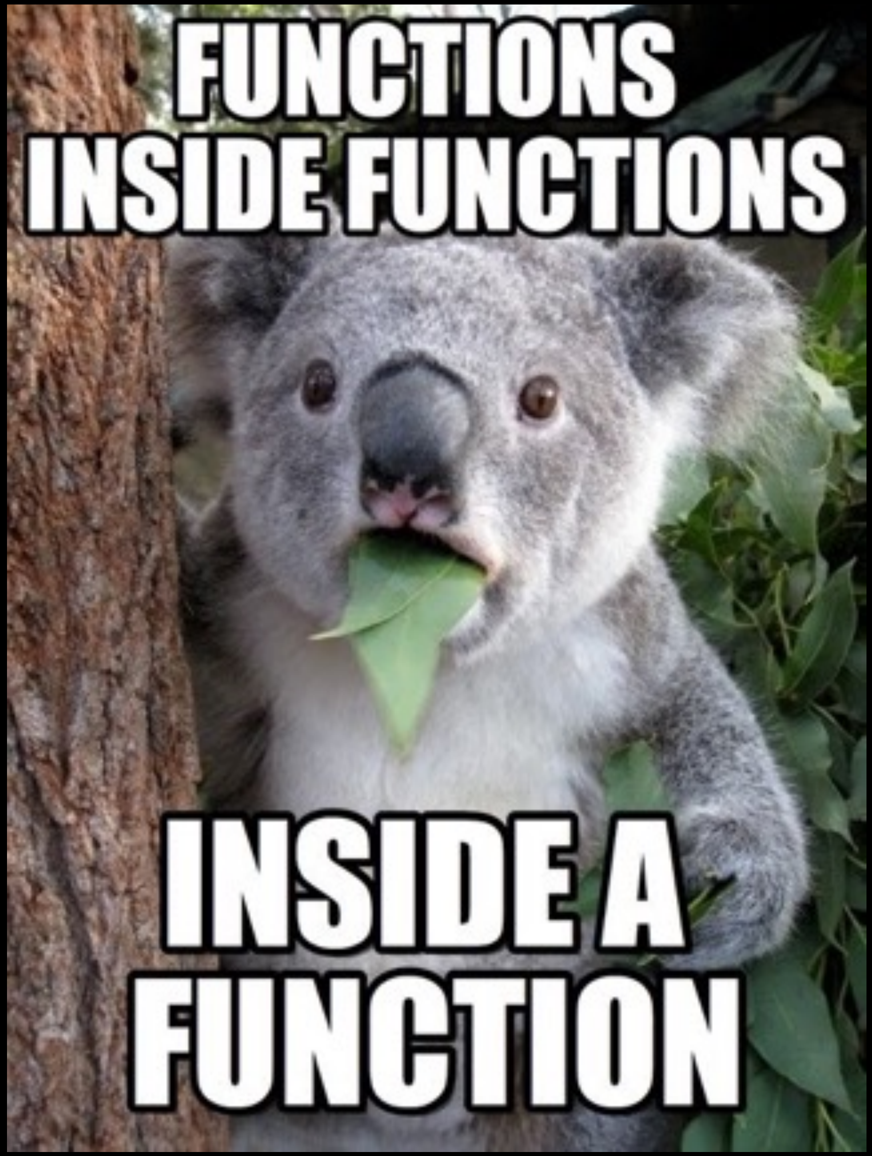
## Functions



# Functions

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

## Functional Swift



## 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 Bridging

## Swift and Objective-C Bridging

- You can mix Swift and Objective-C code in the same project

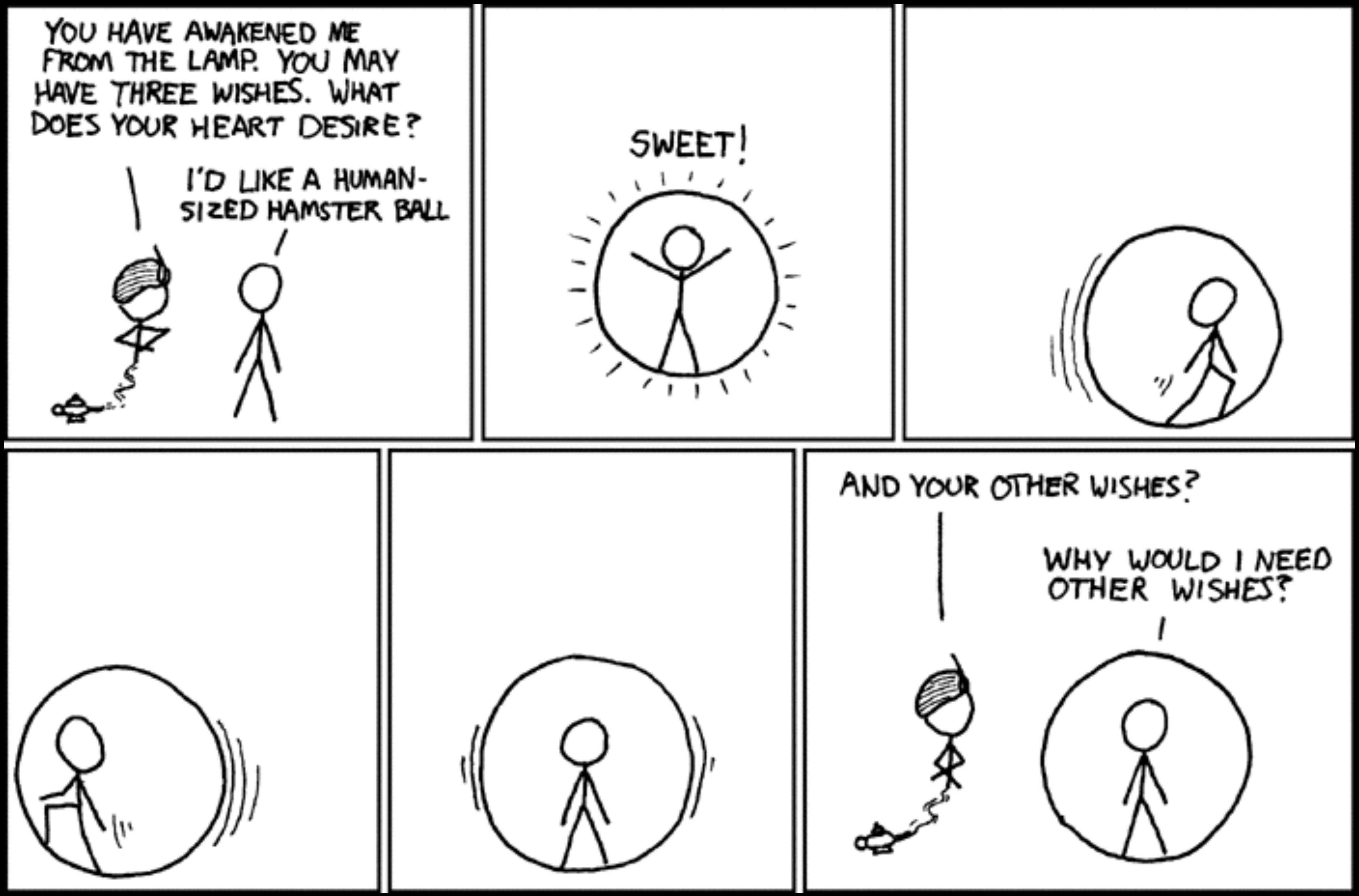
- Xcode can help set up header files and project settings to facilitate interoperability

- 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


YOU HAVE AWAKENED ME FROM THE LAMP. YOU MAY HAVE THREE WISHES. WHAT DOES YOUR HEART DESIRE?
I'D LIKE A HUMAN-SIZED HAMSTER BALL
SWEET!
AND YOUR OTHER WISHES?
WHY WOULD I NEED OTHER WISHES?

## 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

# Resources

- A Swift Tour
- Swift Blog
- Swift Resources
- Functional Programming in Swift
- Swift Tutorials
- This talk

## Me

Jamie Forrest
jamie@jamieforrest.com
@jamieforrest