Malaga Mobile meetup

Value types in Swift: what are they and why should you care?

Adrian Tineo

27 February 2019

Before Swift

Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. It was the main programming language supported by Apple for the macOS and iOS operating systems, and their respective application programming interfaces (APIs) Cocoa and Cocoa Touch until the introduction of Swift.

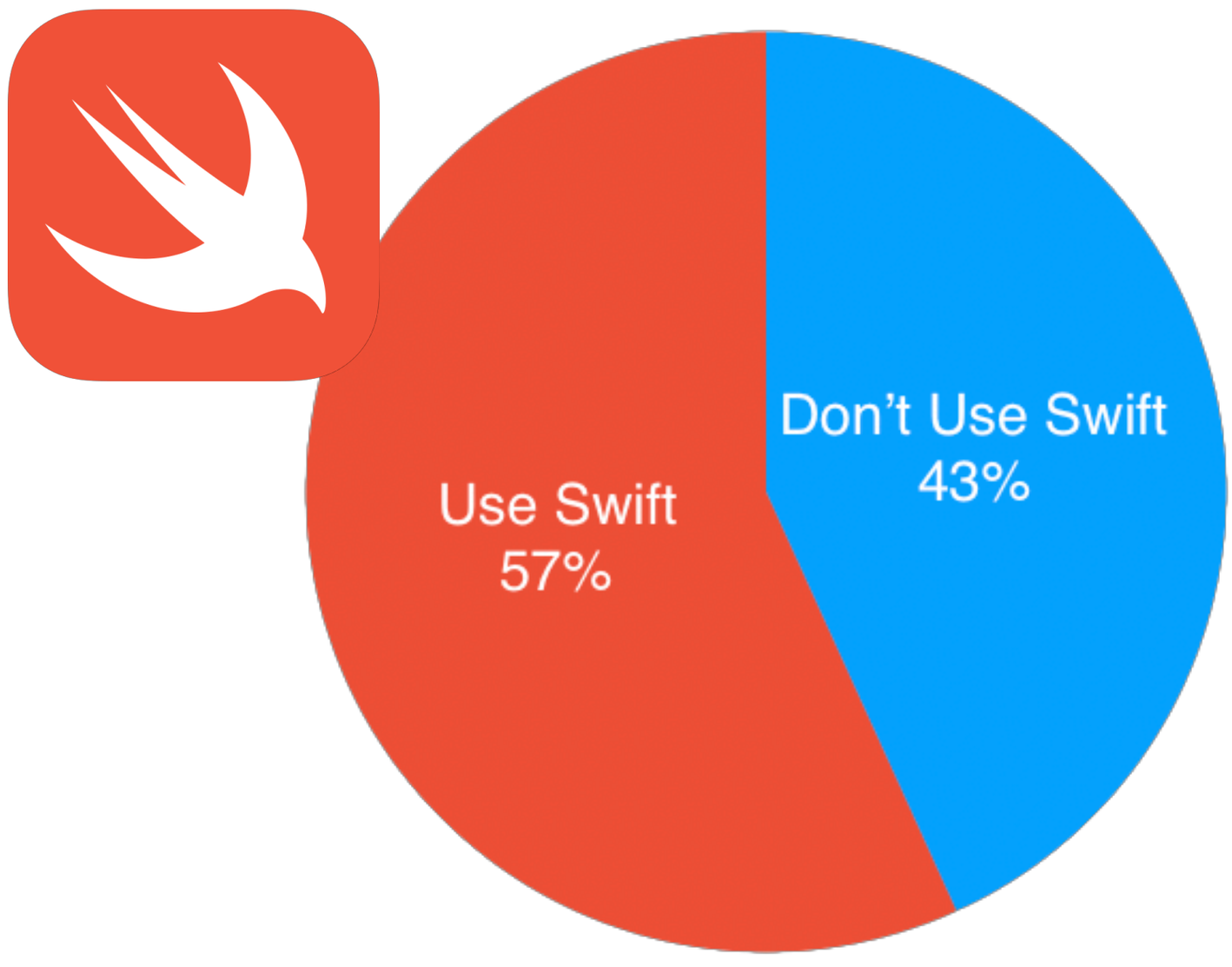
https://en.wikipedia.org/wiki/Objective-C

Brief history of Swift

- July 2010: started by Chris Lattner
- June 2014: first publicly released app written in Swift
- October 2014: Swift 1.1 released to the iOS developer community with Xcode 6.1
- December 2015: Swift is open sourced with home at swift.org and discussion at apple.github.io/swift-evolution/
- June 2016: Swift playgrounds for iPad presented
- Q1 2019: Swift 5.0 to be released

Apps using Swift in 2019

79 non-game apps from the 110 top apps in the App Store



https://blog.andrewmadsen.com/post/182862756395/how-many-apps-use-swift-in-2019

“Swiftiness”

- Swift is more than a prettier syntax over Objective-C

Block declaration and use in Objective-C

```
double (^multiplyTwoValues)(double, double) = ^(double firstValue, double secondValue) {
        return firstValue * secondValue;
};

double result = multiplyTwoValues(2,4);
```

Closure declaration and use in Swift

```
let multiplyTwoValues = { (firstValue: Double, secondValue:Double) -> Double in
    return firstValue*secondValue
}

let result = multiplyTwoValues(2,4)
```

“Swiftiness”

- Swift has a different philosophy as a language
    - Safe, fast, expressive

Safe. The most obvious way to write code should also behave in a safe manner. **Undefined behavior is the enemy of safety, and developer mistakes should be caught before software is in production.** Opting for safety sometimes means Swift will feel strict, but we believe that clarity saves time in the long run.

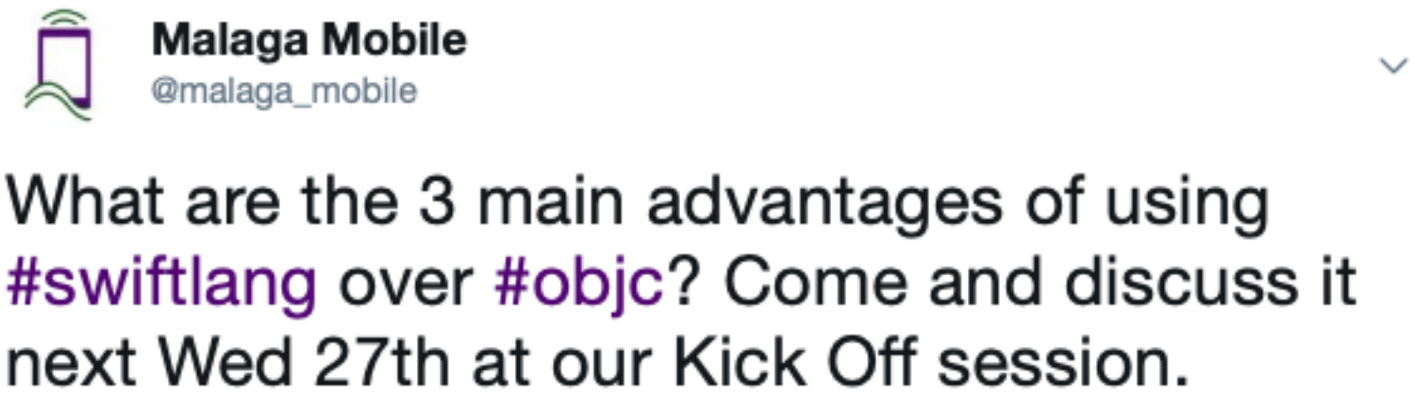
https://swift.org/about/

“Swiftiness”

- API design guidelines
    - Clarity at the point of use is your most important goal
    - Clarity is more important than brevity
    - Write a documentation comment for every declaration

https://swift.org/documentation/api-design-guidelines/

Top 3 Swift features



Top 3 Swift features

- Optionals
- Protocol extensions
- Value types

Top 3 Swift features

- Optionals
- Protocol extensions
- Value types

What's your top 3? Let us know!

What are value types?

> A *value type* is a type whose value is *copied* when it's assigned to a variable or constant, or when it's passed to a function.

> Unlike value types, *reference types* are *not* copied when they are assigned to a variable or constant, or when they are passed to a function. Rather than a copy, a reference to the same existing instance is used.

Reference vs. value types example

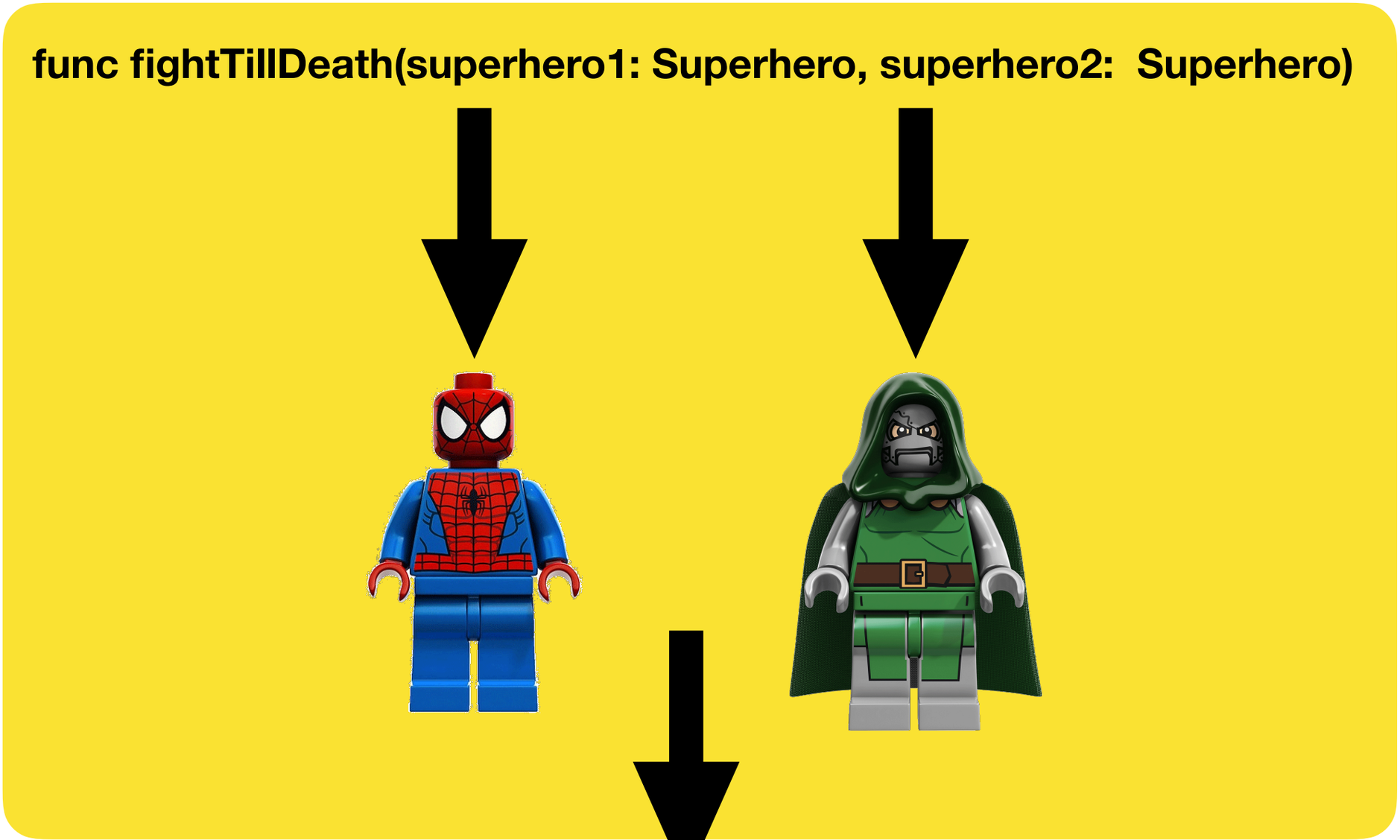
Superhero

name
superpower
health

Character

name
meet(Superhero)

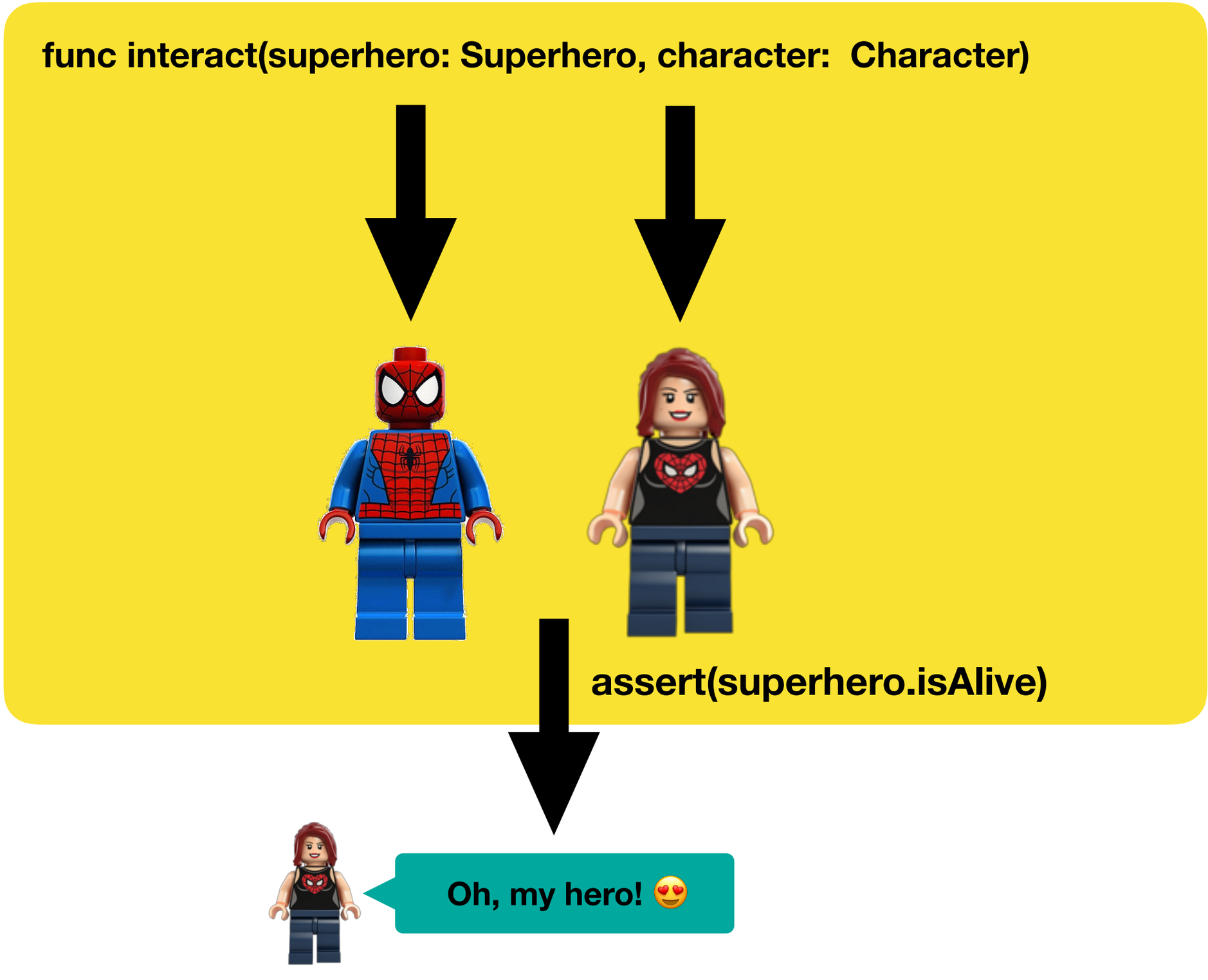
Reference vs. value types example



Alive and kicking!

Alive and kicking!

Reference vs. value types example



Reference type (class)

func fightTillDeath(superhero1: Superhero, superhero2: Superhero)

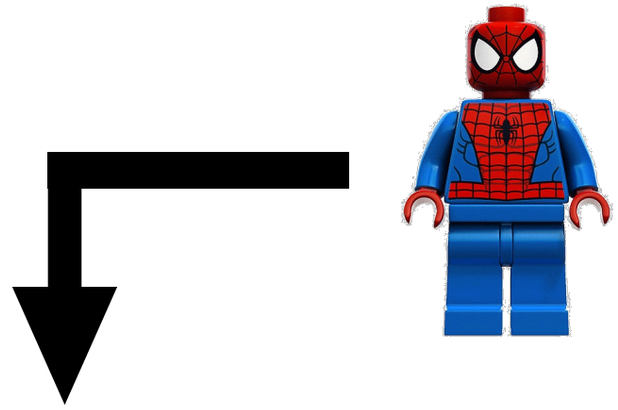
Alive and kicking!

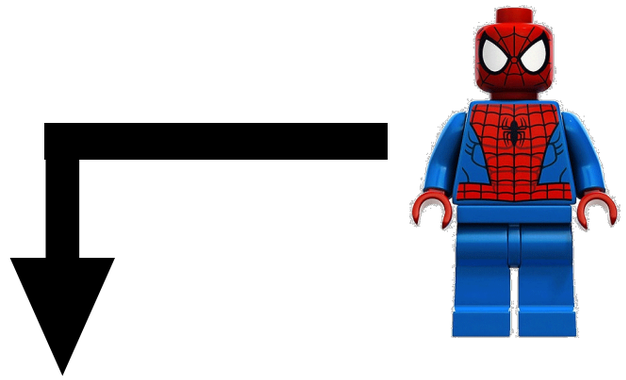
func interact(superhero: Superhero, character: Character)

assert(superhero.isAlive)

superhero_reference_type.playground

Value type (struct)

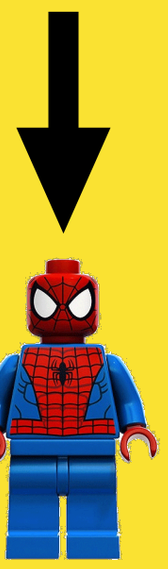




func fightTillDeath(superhero1: Superhero, superhero2: Superhero)

Alive and kicking!

func interact(superhero: Superhero, character: Character)



assert(superhero.isAlive)

Oh. my hero! 😍

superhero_value_type.playground

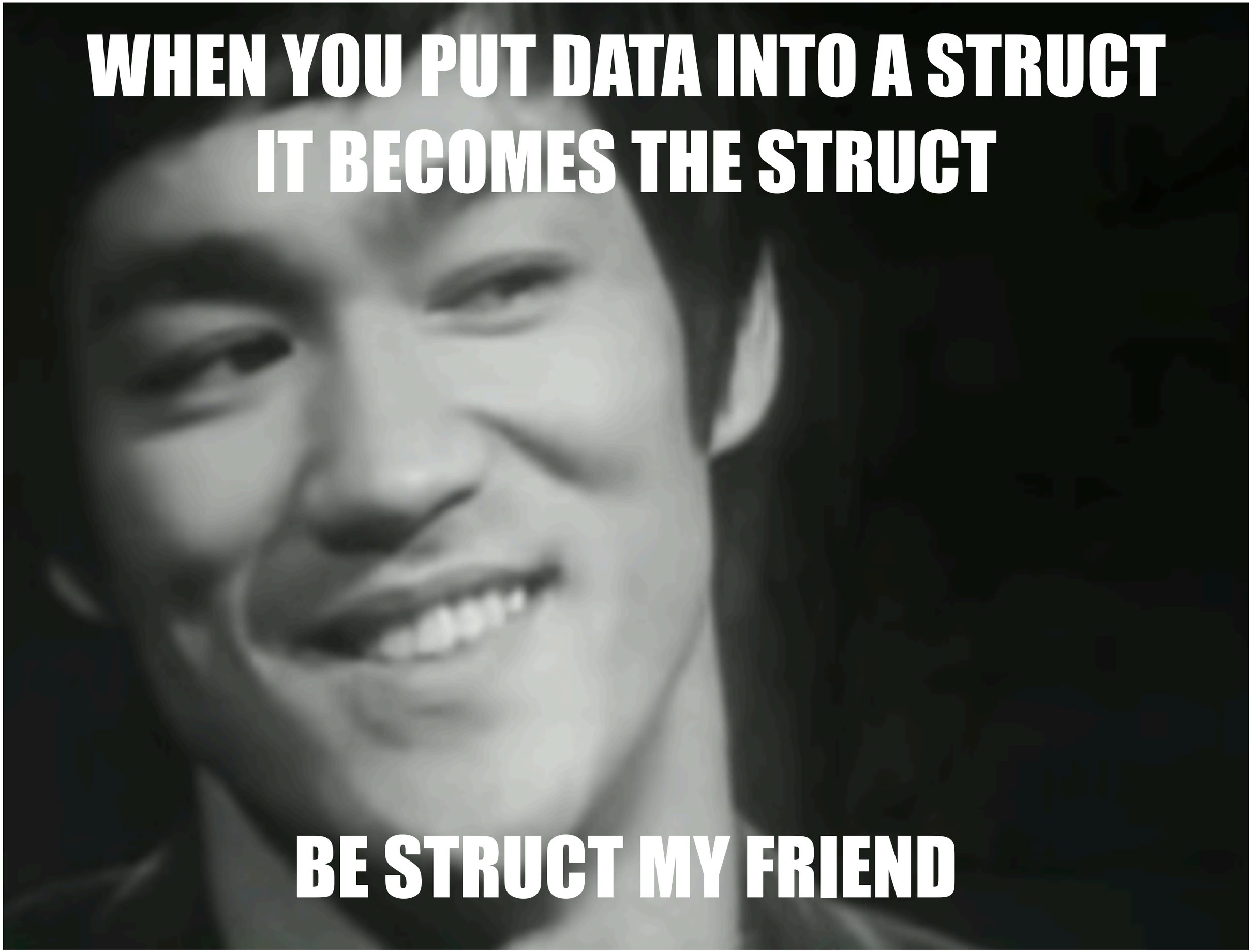
Diff

diff_reference_type_vs_value_type.txt

```
20c20
< class Superhero {
---
> struct Superhero {
31c31
<     func fight(_ enemy:Superhero, luckyFactor: Int) {
---
>     mutating func fight(_ enemy:Superhero, luckyFactor: Int) {
62c62
< func fightTillDeath(superhero1: Superhero, superhero2: Superhero) {
---
> func fightTillDeath(superhero1: inout Superhero, superhero2: inout Superhero) {
93c93
< fightTillDeath(superhero1: spiderman, superhero2: drDoom)
---
> fightTillDeath(superhero1: &spiderman, superhero2: &drDoom)
```

Value types

- Make it easier to reason about code
- Limit rippling effects of modified state
- Explicit about mutable content with qualified keywords (mutating, inout, &)


WHEN YOU PUT DATA INTO A STRUCT
IT BECOMES THE STRUCT
BE STRUCT MY FRIEND