OBJECTIVE-C, MEET SWIFT

HOW TO INTRODUCE SWIFT INTO AN OBJECTIVE-C CODEBASE

INTRODUCTION

Jake Carter

- Software Engineer, Omni Group – 2011 - Present
- Instructor, UW – 2014 - 2016
- Software Engineer, RogueSheep – 2008 - 2011

Agenda

- Why Swift?
  - My Favorite Features
- HOWTO: Swift
  - Adding Swift to an Objective-C App & Framework

WHY SWIFT?

MY FAVORITE FEATURES

Overview

- High Performance & High Productivity
- Modern: Multiple Return Values (Tuples), Optional Arguments, Closures, Generics, Type Inference
- Safe: No uninitialized data, Promotes immutability, Array bounds checks, Integer overflow checks, Raw pointers marked "unsafe"
- Fast: ARM & x86-64 native, Tuned native collections, Swift-specific optimizer, C-like procedural performance

My Favorite Features

- Mutability
  - Value Type vs Reference Type
  - let vs var
- Optionals
  - To nil or not to nil, that is the question
- Generics
  - Strongly typed collections

MUTABILITY

Mutability

- Changing the object in a variable
- Changing the state of an object

Value Type vs Reference Type

- Value Types
    - Struct, Enum
- Reference Types
    - Class, Closure, @objc, *id*

Constants vs Variables

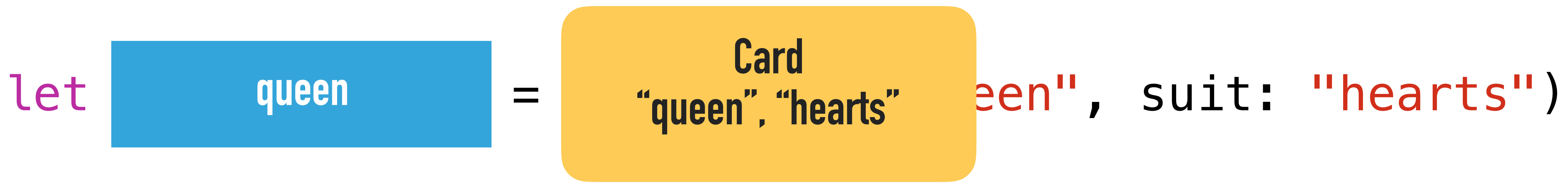
```
let constant: Type = value

var variable: Type = initial value
```

let vs var: Value Type

```swift
struct Card {
    var rank: String
    var suit: String
}
```

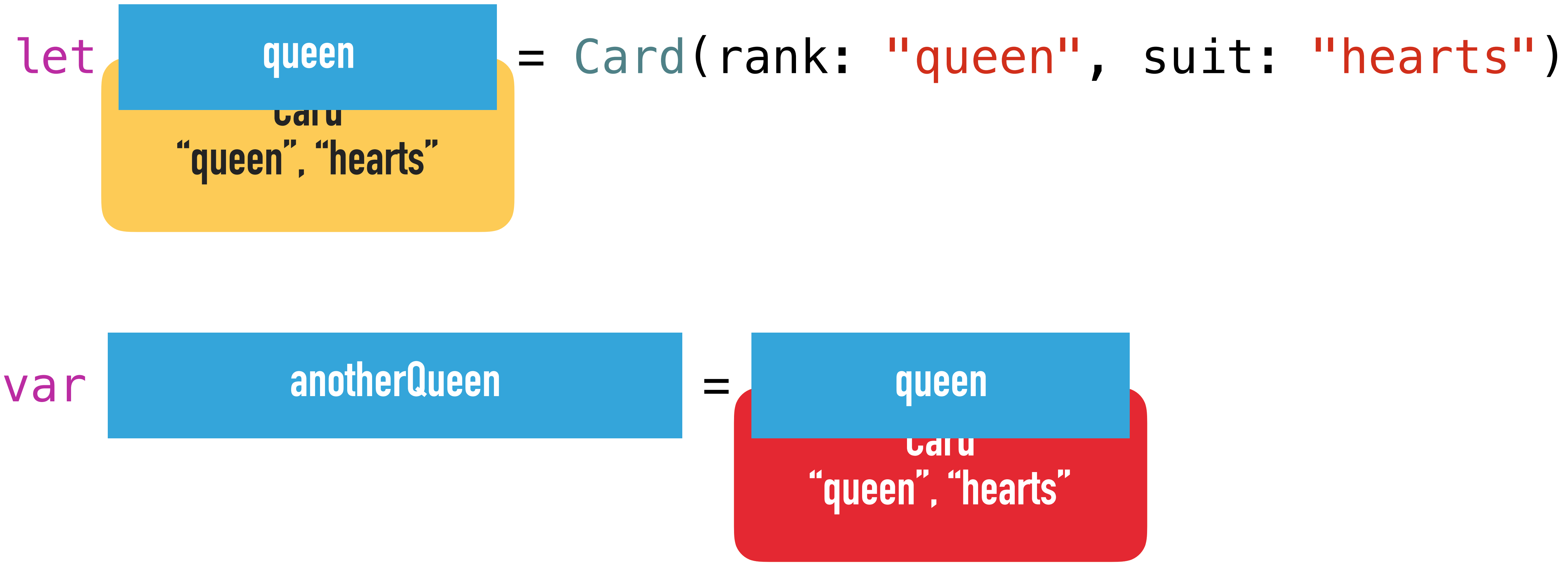
let vs var: Value Type

```
let queen = Card(rank: "queen", suit: "hearts")
var anotherQueen = queen
anotherQueen.suit = "diamonds"
```



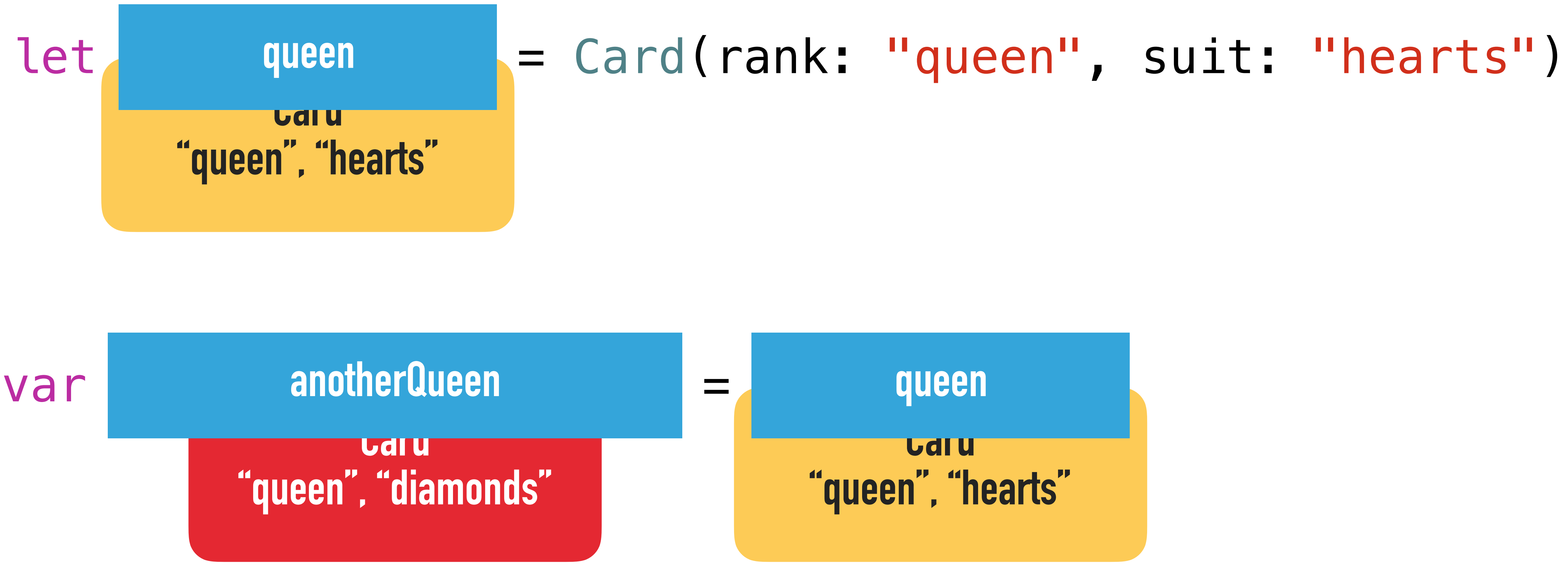
let vs var: Value Type

```swift
let queen = Card(rank: "queen", suit: "hearts")
var anotherQueen = queen
anotherQueen.suit = "diamonds"
```



let vs var: Value Type

```
let queen = Card(rank: "queen", suit: "hearts")
var anotherQueen = queen
anotherQueen.suit = "diamonds"
```

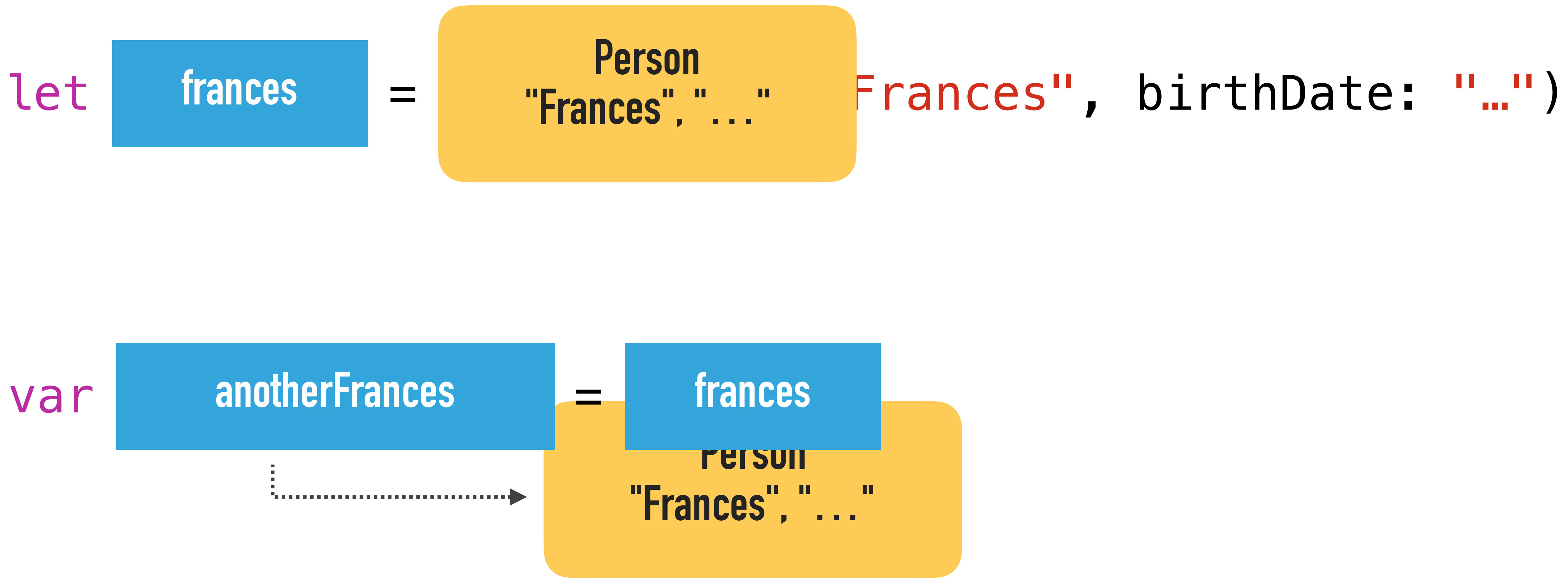


let vs var: Reference Type

```
class Person {
    var name: String
    var birthDate: String
}
```

Reference Types

```
let frances = Person(name: "Frances", birthDate: "01/01/1983")
var anotherFrances = frances
anotherFrances.birthDate = "01/02/1983"
```



let vs var

	let	var
Value Type	Cannot change object; Cannot mutate state	Can change object; Can mutate state*
Reference Type	Cannot change object; Can mutate state	Can change object; Can mutate state

*Functions that self-change Value Type must be marked *mutating*.

Value Type vs Reference Type

```swift
struct Card {
    var rank: String
    var suit: String
}

class Person {
    var name: String
    var birthDate: String
}
```

OPTIONALS

Non-Optional Type

```
var name: String = “Margaret”
name = nil
```

NIL CANNOT BE ASSIGNED TO TYPE 'STRING'

Optional Type

```
var name: String? = “Margaret”
name = nil
```

Optional Type

```
var name: String? = “Margaret”
name = nil

let chars = name.characters
```

VALUE OF OPTIONAL TYPE 'STRING?' NOT UNWRAPPED;

Optional Binding

```
var name: String? = “Margaret”
name = nil

if let name = name {
    let chars = name.characters
}
```

Optional Chaining

```swift
var name: String? = “Margaret”
name = nil

if let name = name {
    let chars = name.characters
}

let chars = name?.characters
```

Optional Binding vs Optional Chaining

```
var name: String? = “Margaret”
name = nil

if let name = name {
    let chars = name.characters
}
```

let chars: CharacterView

```
let chars = name?.characters
```

let chars: CharacterView?

GENERICS

Generic Collections

```
struct Array<Element> { … }

struct Dictionary<Key: Hashable, Value> { … }
```

Array

```
let names: Array<String> = ["Foo", … ]
```

Array

```
let names: Array<String> = ["Foo", … ]

let names: [String] = ["Foo", … ]
```

Array

```
let names: Array<String> = ["Foo", … ]

let names: [String] = ["Foo", … ]

let names = ["Foo", … ]
```

Dictionary

```
let nameAge: Dictionary<String, Int> = ["Pam" : 36, … ]

let nameAge: [String : Int] = ["Pam" : 36, … ]

let nameAge = ["Pam" : 36, … ]
```

HOWTO: SWIFT

ADDING SWIFT TO AN OBJECTIVE-C APP & FRAMEWORK

DEMO

Demo Wrap Up

- Added Swift to Objective-C App Target
    - Bridging Header, Enabled Module Support, Subclassed Objective-C class in Swift
- Swiftified Objective-C Framework Headers
    - Nullability Annotations, Typed Collections
- Added Swift to Objective-C Framework Target
    - NO Bridging Header/Must use Umbrella Header, Enabled Module Support, Extended Objective-C class in Swift
- Utilized Framework Swift in App

Bridging Headers (From same App Target)

	Import into Swift	Import into Objective-C
Swift	No import statement	#import "ProductModuleName-Swift.h"
Objective-C	No import statement; Objective-C **bridging** header required	#import "Header.h"

Apple Inc. "Using Swift with Cocoa and Objective-C (Swift 3)." iBooks. https://itun.es/us/1u3-0.l

Bridging Headers (From same Framework Target)

	Import into Swift	Import into Objective-C
Swift	No import statement	#import <ProductName/ProductModuleName-Swift.h>
Objective-C	No import statement; Objective-C **umbrella** header required	#import "Header.h"

Apple Inc. "Using Swift with Cocoa and Objective-C (Swift 3)." iBooks. https://itun.es/us/1u3-0.l

Importing Frameworks

	Import into Swift	Import into Objective-C
Any language framework	import FrameworkName	@import FrameworkName;

Apple Inc. “Using Swift with Cocoa and Objective-C (Swift 3).” iBooks. https://itun.es/us/1u3-0.l

THANK YOU

@JakeCarter

AverageJake.com