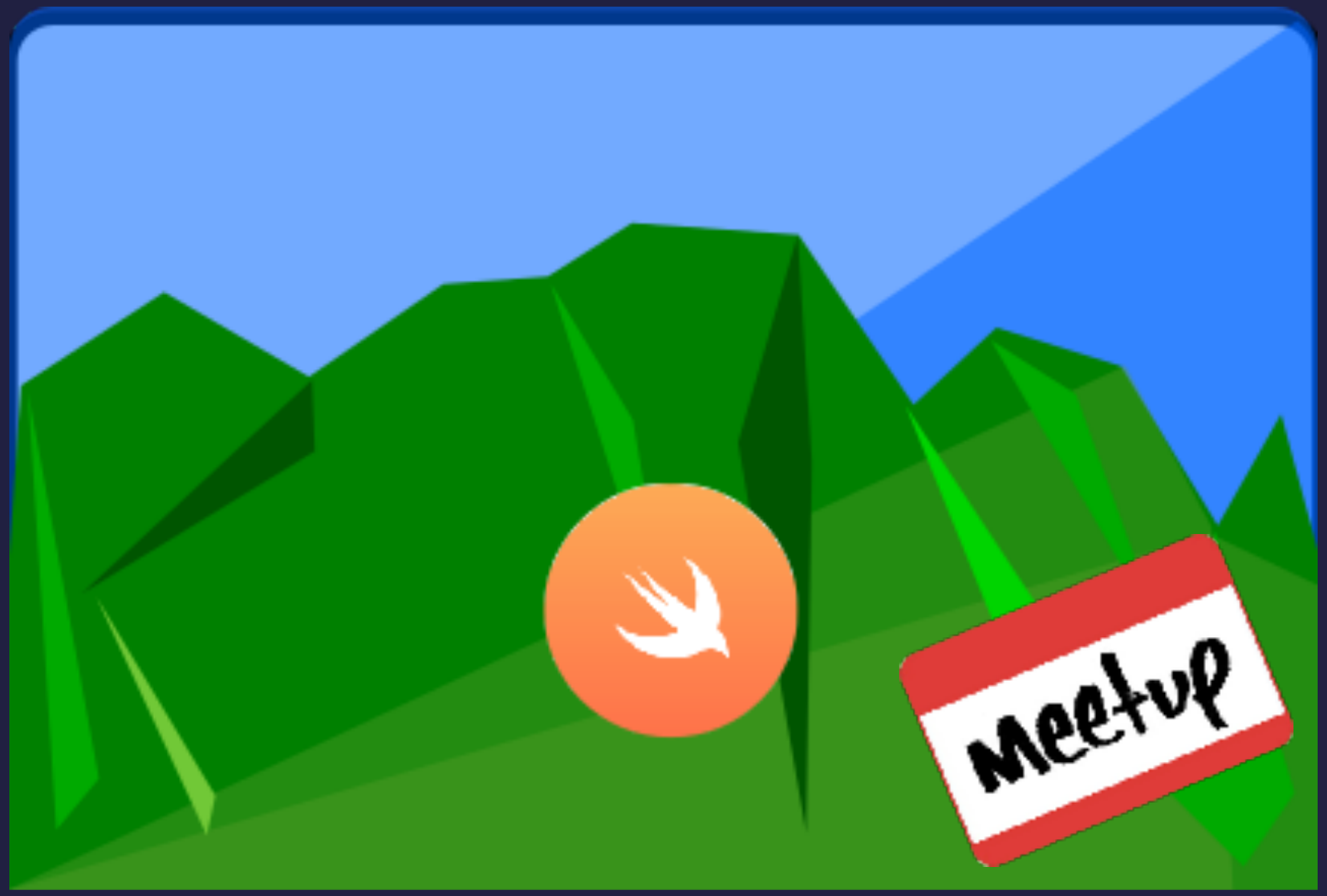


CARACAS SWIFT MEETUP

#ccsSwiftMeetup

Primera Reunión!

First Meeting!

Agenda del Evento

#ccsSwiftMeetup

- SWIFT, ¿SWIFT?
- COMPOSICIÓN FUNCIONAL
- CASO DE ESTUDIO 1
- Q&A

Event Agenda

- SWIFT, SWIFT?
- FUNCTIONAL COMPOSITION
- REAL LIFE CASE 1
- Q&A


Caracas Swift Meetup

¿Quienes Somos?

#ccsSwiftMeetup

- INGENIERO DE SISTEMAS
- IOS DEVELOPER DESDE 2010
- CSA & COFOUNDER @CODEFUEL
- APRENDIENDO SWIFT DESDE JUNIO 2014

Who are we?

- SYSTEMS ENGINEER
- IOS DEVELOPER SINCE 2010
- CSA & COFOUNDER @CODEFUEL
- LEARNING SWIFT SINCE JUN 2014

#ccsSwiftMeetup

SWIFT, Swift?

Caracas Swift Meetup

Features / Características

- OPTIONAL VALUES
- TYPE SAFE
- TYPE INFERENCE
- VALUE AND REFERENCE TYPES
- GENERICS
- ENUMS
- CLOSURES
- FUNCTIONS ARE FIRST CLASS CITIZENS
- FUNCTIONAL COMPOSITION
- MANY MANY MORE ... (COMPUTED PROPERTIES, LAZY INITIALISATION, PROTOCOLS, ETC)

Optionals

"THE CONCEPT OF OPTIONALS DOESN'T EXISTS IN C OR OBJECTIVE-C. THE NEAREST THING IN OBJECTIVE-C IS THE ABILITY TO RETURN NIL FROM A METHOD THAT WOULD OTHERWISE RETURN AN OBJECT, WITH NIL MEANING "THE ABSENCE OF A VALID OBJECT""

(THE SWIFT PROGRAMMING LANGUAGE BOOK)

"EL CONCEPTO DE LOS VALORES OPTIONALES NO EXISTE EN C O EN OBJECTIVE-C. LO MÁS CERCANO A ESTE CONCEPTO EN OBJECTIVE-C IS LA POSIBILIDAD DE DEVOLVER NIL EN UN MÉTODO QUE DE OTRA FORMA RETORNARÍA UN OBJETO, DONDE NIL REPRESENTA LA AUSENCIA DE UN OBJECTO VALIDO

(THE SWIFT PROGRAMMING LANGUAGE BOOK)

Optionals

YOU USE OPTIONALS IN SITUATIONS WHERE A VALUE MAY BE ABSENT. AN OPTIONAL SAYS:

- THERE IS A VALUE, AND IT EQUALS X

OR

- THERE ISN'T A VALUE AT ALL (NIL)

USAREMOS OPCIONALES EN SITUACIONES EN LAS CUALES EL VALOR PUEDE ESTAR AUSENTE. LOS OPCIONALES NOS DICEN:

- EXISTE UN VALOR Y EL VALOR ES ... X

Ó

- NO HAY NINGÚN VALOR (NIL)

Optionals


```
var x:Int? //<= Optional
var y:Int = 5 //Non Optional
var z:Int = 0 //Non Optional
```

Optionals

```
Int? != Int
```

Type Safe

#ccsSwiftMeetup

"SWIFT HELPS YOU TO BE CLEAR ABOUT THE TYPES OF VALUES YOUR CODE CAN WORK WITH. IF PART OF YOUR CODE EXPECTS A STRING, TYPE SAFETY PREVENTS YOU FROM PASSING IT AN INT BY MISTAKE."
(THE SWIFT PROGRAMMING LANGUAGE BOOK)

"SWIFT NOS AYUDA A SER CLAROS CON RESPECTO AL TIPO DE VALORES QUE NUESTRO CÓDIGO PUEDE MANEJAR. SI ALGÚN FRAGMENTO DEL CÓDIGO ESPERA UN VALOR DE TIPO STRING, EL COMPILADOR NO PERMITIRÁ RECIBIR UN TIPO INT POR ERROR"
(THE SWIFT PROGRAMMING LANGUAGE BOOK)


Caracas Swift Meetup

Type Safe


```
let slide:String = "🍕"
let pizza:String = " Pizza!"
let pizzaSlide:String = slide + pizza
let pizzaQuarter:String = 0.25
```

```
Cannot convert value of Type ‘Double’ to
specified type ‘String’
```

Type Inference

"TYPE INFERENCE IS PARTICULARLY USEFUL WHEN YOU DECLARE A CONSTANT OR VARIABLE WITH AN INITIAL VALUE. THIS IS OFTEN DONE BY ASSIGNING A LITERAL VALUE (OR LITERAL) TO THE CONSTANT OR VARIABLE AT THE POINT THAT YOU DECLARE IT. (A LITERAL VALUE IS A VALUE THAT APPEARS DIRECTLY IN YOUR SOURCE CODE, SUCH AS 42 AND 3.14159 IN THE EXAMPLES BELOW.)"

(THE SWIFT PROGRAMMING LANGUAGE BOOK)

"LA INFERENCIA DE TIPOS ES PARTICULARMENTE PRACTICA CUANDO DECLARAMOS UNA VARIABLE O UNA CONSTANTE CON UN VALOR INICIAL. PARA ELLO ASIGNAMOS UN VALOR LITERAL A LA CONSTANTE O VARIABLE AL MOMENTO DE DECLARARLA. (UN VALOR LITERAL ES AQUEL QUE APARECE DIRECTAMENTE EN EL CÓDIGO FUENTE, COMO POR EJEMPLO 42 Y 3.14159 EN EL EJEMPLO "

(THE SWIFT PROGRAMMING LANGUAGE BOOK)

Type Inference


```
let meaningOfLife = 42
let pi = 3.14159

_stdlib_getDemangledTypeName(pi)
_stdlib_getDemangledTypeName(meaningOfLife)
```

```
42
3.14159

“Swift.Double"
"Swift.Int"
```


Value and reference Types

#ccsSwiftMeetup

"TYPES IN SWIFT FALL INTO ONE OF TWO CATEGORIES: FIRST, "VALUE TYPES", WHERE EACH INSTANCE KEEPS A UNIQUE COPY OF ITS DATA, USUALLY DEFINED AS A `struct`, `enum`, OR TUPLE. THE SECOND, "REFERENCE TYPES", WHERE INSTANCES SHARE A SINGLE COPY OF THE DATA, AND THE TYPE IS USUALLY DEFINED AS A `class`."

(APPLE'S SWIFT BLOG)

"EN SWIFT LOS TIPOS PERTENECEN EN UNA DE DOS CATEGORIAS: PRIMERO, "TIPO VALORES", DONDE CADA INSTANCIA MANTIENE UNA COPIA ÚNICA DE SU DATA, USUALMENTE DEFINIDA COMO UN `struct`, `enum`, O UNA TUPLA. LA SEGUNDA "TIPO REFERENCIAS", DONDE LAS INSTANCIAS COMPARTEN UNA SOLA COPIA DE LA DATA Y EL TIPO ES DEFINIDO COMO UNA CLASE "

(THE SWIFT PROGRAMMING LANGUAGE BOOK)

Value and reference Types

#ccsSwiftMeetup

```swift
//Value Type
struct padawan {
    var name:String
}

var anakin = padawan(name: "Anakin")
var azoka = anakin

anakin.name = "Darth Vader"
anakin.name
azoka.name
```

```swift
//Reference Type
class sithLod {
    var ligthabreModel:String?
}

var dartMoul = sithLod()
dartMoul.ligthabreModel = "Double"

var duku = dartMoul
duku.ligthabreModel = "Curved"

dartMoul.ligthabreModel
duku.ligthabreModel
```


Caracas Swift Meetup

Generics

"GENERIC CODE ENABLES YOU TO WRITE FLEXIBLE, REUSABLE FUNCTIONS AND TYPES THAT CAN WORK WITH ANY TYPE..... YOU CAN WRITE CODE THAT AVOIDS DUPLICATION AND EXPRESSES ITS INTENT IN A CLEAR, ABSTRACTED MANNER.."

(THE SWIFT PROGRAMMING LANGUAGE BOOK)

"EL CÓDIGO GENERO PERMITE ESCRIBIR FUNCIONES REUSABLES Y FLEXIBLES QUE PUEDEN TRABAJAR CON CUALQUIER TIPO DE DATO .... PUEDES ESCRIBIR CÓDIGO QUE PREVENGA LA DUPLICACIÓN Y EXPRESE SU INTENCIÓN DE MANERA CLARA Y ABSTRAÍDA "

(THE SWIFT PROGRAMMING LANGUAGE BOOK)

Generics


```
func transform<A,B> (x:A, f:A->B) -> B {
    return f(x)
}

let trans1 = transform(5, f: {x in x + 2 })
let trans2 = transform("Dead ", f: {x in x + "Star 🌐" })
```

```
7
"Dead Star 🌐"
```

Enums

#ccsSwiftMeetup

"AN ENUMERATION DEFINES A COMMON TYPE FOR A GROUP OF RELATED VALUES AND ENABLES YOU TO WORK WITH THOSE VALUES IN A TYPE-SAFE WAY WITHIN YOUR CODE."
(THE SWIFT PROGRAMMING LANGUAGE WEBSITE)

"UN NUMERATIVO SE DEFINE COMO UN GRUPO DE VALORES RELACIONADOS CON EL MISMO TIPO QUE PERMITE TRABAJAR CON ESOS VALORES DE MANERA SEGURA EN EL CÓDIGO"
(THE SWIFT PROGRAMMING LANGUAGE WEBSITE)


Enums


```
enum Planet {
    case Tatoine
    case Mustafa
    case Alderan
    case Corusant
    case Naboo
    case Endor
}

let capitalPlanet = Planet.Corusant

enum Force {
    case lightSide(padawan)
    case darkSide(Bool, Bool)
}

let myForceChooise = Force.lightSide(anakin)

switch myForceChooise {
    case .lightSide(let jedi):
        print(jedi.name)
    case .darkSide(true, true) :
        print("You don't know the power of the dark side")
```

Closures

#ccsSwiftMeetup

"CLOSURES ARE SELF-CONTAINED BLOCKS OF FUNCTIONALITY THAT CAN BE PASSED AROUND AND USED IN YOUR CODE. CLOSURES IN SWIFT ARE SIMILAR TO BLOCKS IN C AND OBJECTIVE-C AND TO LAMBDAS IN OTHER PROGRAMMING LANGUAGES."

(THE SWIFT PROGRAMMING LANGUAGE WEBSITE)

"LOS "CIERRES" SON BLOQUES DE FUNCIONALIDAD AUTO-CONTENIDOS QUE PUEDEN SER REFERENCIADOS Y USADOS EN EL CÓDIGO. LOS "CIERRES" EN SWIFT SON SIMILARES A LOS BLOQUES EN C Y OBJECTIVE-C Y LOS LAMBDAS EN OTROS LENGUAJES DE PROGRAMACIÓN"

(THE SWIFT PROGRAMMING LANGUAGE WEBSITE)


Caracas Swift Meetup

Closures


```
var buzzLighsaber = { return "Buzzz" }
var lightSaberColor:(String) -> (String) = {  x in x + " Saber" }
var myLightSaber = lightSaberColor("Blue")
```

Functions are First Class citizens

#ccsSwiftMeetup

"IN SWIFT, A FUNCTION IS A FIRST-CLASS CITIZEN. THIS MEANS THAT A FUNCTION CAN BE USED WHEREVER A VALUE CAN BE USED. FOR EXAMPLE, A FUNCTION CAN BE ASSIGNED TO A VARIABLE; A FUNCTION CAN BE PASSED AS AN ARGUMENT IN A FUNCTION CALL; A FUNCTION CAN BE RETURNED AS THE RESULT OF A FUNCTION"

(O'REILLY IOS 8 PROGRAMMING FUNDAMENTALS WITH SWIFT)

"EN SWIFT, UNA FUNCIÓN ES UN CIUDADANO DE PRIMERA CLASE. ESTO SIGNIFICA QUE UNA FUNCIÓN PUEDE SER USADA EN CUALQUIER SITIO DONDE SE PUEDA USAR UN VALOR. POR EJEMPLO, UNA FUNCIÓN PUEDE SER ASIGNADA A UNA VARIABLE; UNA FUNCIÓN PUEDE SER PASADA COMO ARGUMENTO EN UNA LLAMADA A UNA FUNCIÓN; UNA FUNCIÓN PUEDE SER RETORNADA COMO RESULTADO DE UNA LLAMADA DE OTRA FUNCIÓN"

(O'REILLY IOS 8 PROGRAMMING FUNDAMENTALS WITH SWIFT)

Functions are First Class citizens


```
func transform<A,B>(x:A, f:A->B) -> B {
    return f(x)
}

var lightSaberColor:(String) -> (String) = {  x in x + "
Saber" }
```

```
func activateDroid(x:String) -> (String) -> Bool {
    return { y in x == "BB8" && y == "Active" }
}

let droid = activateDroid("BB8")
```


Functional Composition

$$y = f(x),\ z = g(y)$$

$x \text{ -> } f(x) \text{ -> } y$ $\qquad$ $y \text{ -> } g(y) \text{ -> } z$

$$z = g(f(x))$$

Functional Composition

#ccsSwiftMeetup

```swift
//Functional Composition

func buildDeadStar(part:String, number:Int) -> [Int:String] {
    return [number: part]
}

func deadStarStock(dict:[Int:String]) -> Bool {
    for (key, value) in dict {
        if key == 0  || value == ""{
            return false
        }
    }
    return true
}


var deadStarValidParts = deadStarStock(buildDeadStar("Planet Green Laser", number: 66))
```


Caracas Swift Meetup

Case of Study

Live coding

Sources

- SWIFT.ORG
- HTTPS://DEVELOPER.APPLE.COM/SWIFT/BLOG/
- HTTPS://DEVELOPER.APPLE.COM/VIDEOS/WWDC2015/
- HTTPS://DEVELOPER.APPLE.COM/LIBRARY/
- THE SWIFT PROGRAMMING LANGUAGE BOOK
- FUNCTIONAL PROGRAMMING IN SWIFT BOOK

Sources

FOLLOW

- @NATASHATHEROBOT
- @ANDY_MATUSCHAK
- @RWENDERLICH
- @CHRISEIDHOF

Contact!

#ccsSwiftMeetup

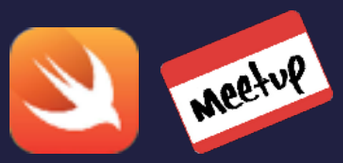
KEEP IN TOUCH!

- @JORGEVMENDOZA
- @CODEFUEL

- CODEFUEL.ME
- THESWIFT.NINJA

- HTTPS://GITHUB.COM/CARACASSWIFT

CODEFUEL



Caracas Swift Meetup

THANKS! / GRACIAS!

#ccsSwiftMeetup

Caracas Swift Meetup