

# Swift

Le nouveau langage, conçu par Apple

# Florian PETIT

- Ingénieur Étude & Développement iOS - Pictime
- DUT - Supinfo
- CocoaHeads
- <http://mrcloud.github.io>
- @Florian\_MrCloud



# Disclaimer

- Introduction / Concepts Globaux + exemples
- Sessions avancées à venir

# Qu'est-ce que Swift ?

- Langage
- Projet initié en 2010 (Apple / LLVM)
- C/Objective-C (sans compatibilité avec C)
- Moderne, Flexible, Simple, Interactif
- “Safe”



# Qu'est-ce que Swift ?

- Bénéficie des améliorations apportées au compilateur et à l'infrastructure de LLVM
- ARC
- S'appuie sur les évolutions de l'Objective-C moderne (blocks, literals, modules)
- S'inspire largement de fonctionnalités disponibles dans d'autres languages (Scala, C# ...)

# “Safe”

- Variables: `var languageName: String = "Swift"`
- Constantes (!immutable): `let isAwesome: Bool = true`
- Un compilateur très peu permissif

```
14 - (void)someMethod {  
15     NSString *languageName = @"Swift";  
16     NSNumber *isAwesome = @(true);  
17  
18     languageName = isAwesome;  
19 }
```

```
11 var languageName = "Swift"  
12 let isAwesome = true  
! 13 languageName = isAwesome
```

```
20 int i = 123;  
21 if (i) {  
22  
23 }
```

```
14  
15 let i = 1  
! 16 if i {  
17     // Oops  
18 }  
19 |
```

# Inférence de type

- `let languageName = "Swift" // inferred as String`
- `var version = 1.0 // inferred as Double`
- `let introduced = 2014 // inferred as Int`
- `let isAwesome = true // inferred as Bool`
- Typé explicitement à la compilation
- `let aValue = 3 + 0.14159`

# Types Usuels

- Int
- String
- Bool
- Float / Double
- Array
- Dictionary



```
struct String { }
```

```
struct Int : SignedInteger { }
```

```
struct Float { }
```

```
struct Bool { }
```

```
struct Dictionary<KeyType : Hashable, ValueType> : Collection,  
DictionaryLiteralConvertible { }
```

```
struct Array<T> : MutableCollection, Sliceable { }
```

# Puissant

- Tuples: groupe de valeurs
- ```
let http404Error = (404, "Not Found")  
let (statusCode, statusMessage) = http404Error
```
- Optionals
- Abus de constantes

# Syntaxe

- Très épurée (Ruby ...)
- Peu permissive ( pas de if sans {}, default obligatoire dans certains switch ...)

# Control Flow

- Demo

# Fonctions

- `func funcName(externalParamName [var/let/inout]  
internalParamName: Type, anotherExPName  
anotherInPName: Type) -> (ReturnType) {}`

```
func join(string s1: String , toString s2: String , withJoiner joiner: String) -> String {}
```

```
join(string: "hello", toString: "world", withJoiner: ", ")  
join(string: "hello", toString: "world")
```

```
func join(#string: String , #toString: String , #withJoiner: String ) -> String { } //
```

Syntaxe condensée, le nom du parametre est le meme en interne & externe

```
func join(s1: String , s2: String , joiner: String = " ") -> String { } // Swift crée  
automatiquement un nom de paramètre externe pour les valeurs par défaut  
join("hello", "world", joiner: "-")
```

# Closures

- Blocks
- Peuvent capturer et stocker des références
- Gestion de la mémoire par le runtime swift

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]  
var reversed = sort(names) { $0 > $1 }
```

# Enum

- Enums très puissants :
  - Plusieurs types supportés: (String, char, Int, Float)
  - Non homogène (~= unions)
  - Traitées comme des classes.

```
enum Barcode { // non homogene
    case UPCA(Int, Int, Int)
    case QRCode(String)
}
var productBarcode = Barcode.UPCA(8, 85909_51226, 3)
productBarcode = .QRCode("ABCDEFGHJKLMNOP")
switch productBarcode {
case .UPCA(let numberSystem, let identifier, let check):
    println("UPC-A with value of \(numberSystem), \(identifier), \(check).")
case .QRCode(let productCode):
    println("QR code with value of \(productCode).")
}
```

```
enum Planet: Int { // homogene
    case Mercury = 1, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
}
```

# Struct & Classes

- Demo



# Enum, Structs & Classes

| Feature      | Enum  | Struct | Class     |
|--------------|-------|--------|-----------|
| Properties   | ✓     | ✓      | ✓         |
| Methods      | ✓     | ✓      | ✓         |
| Initializers | ✓     | ✓      | ✓         |
| Extensions   | ✓     | ✓      | ✓         |
| Protocols    | ✓     | ✓      | ✓         |
| Subscript    |       | ✓      | ✓         |
| Inheritance  |       |        | ✓         |
| Type Casting |       |        | ✓         |
| Ref Counting |       |        | ✓         |
| Type         | Value | Value  | Reference |

# Propriétés

- Stockés :

```
struct FixedLengthRange {  
    var firstValue: Int  
    let length: Int  
}  
var rangeOfThreeItems = FixedLengthRange(firstValue: 0, length: 3) // mutable  
rangeOfThreeItems.firstValue = 6  
let
```

- Calculées :

```
struct AlternativeRect {  
    var origin = Point()  
    var size = Size()  
    var center: Point {  
        get {  
            let centerX = origin.x + (size.width / 2)  
            let centerY = origin.y + (size.height / 2)  
            return Point(x: centerX, y: centerY)  
        }  
        set {  
            origin.x = newValue.x - (size.width / 2)  
            origin.y = newValue.y - (size.height / 2)  
        }  
    }  
    var area: Double { //readonly  
        return size.width * size.height  
    }  
}
```

# Property Observer

```
class StepCounter {  
    var totalSteps: Int = 0 {  
        willSet(newTotalSteps) {  
            println("About to set totalSteps to \$(newTotalSteps)")  
        }  
        didSet {  
            if totalSteps > oldValue {  
                println("Added \$(totalSteps - oldValue) steps")  
            }  
        }  
    }  
}
```

# Méthodes

```
class Player {
    var tracker = LevelTracker()
    let playerName: String
    func completedLevel(level: Int) {
        LevelTracker.unlockLevel(level + 1)
        tracker.advanceToLevel(level + 1)
    }
    init(name: String) {
        playerName = name
    }
}

class LevelTracker {
    static func levelIsUnlocked(level: Int) -> Bool {
        return level <= highestUnlockedLevel
    }
    var currentLevel = 1
    mutating func advanceToLevel(level: Int) -> Bool {
        if LevelTracker.levelIsUnlocked(level) {
            currentLevel = level
            return true
        } else {
            return false
        }
    }
}
```

# Optionals

- Nouveau concept
- nil d'objective-c signifie: l'absence d'objet valide pour un pointeur (ne fonctionne pas avec les structs, les types C, enums etc) -> nécessité d'utiliser des valeurs spéciales pour représenter ce concept: NSNotFound ...
- Notion d'absence de valeur, qui fonctionne pour tous les types.

```
let possibleNumber = "123"  
let convertedNumber = possibleNumber.toInt()  
// convertedNumber is inferred to be of type "Int?", or "optional Int"
```

# Optionals

- Forced Unwrapping (déballage forcé ?)

```
if convertedNumber {  
    println("\(possibleNumber) has an integer value of \((convertedNumber!))")  
}
```

- Optional Binding

```
if let actualNumber = possibleNumber.toInt() {  
    println("\(possibleNumber) has an integer value of \((actualNumber)")  
}
```

- Implicit Unwrapping

```
let assumedString: String != "An implicitly unwrapped optional string."  
println(assumedString)
```

# Type Casting

- Class A {}
- Class B: A {}
- Class C: A {}
- let array = [B(..), C(..), B(..)] // inferré en A[]
- Type check avec le mot clé “is”
- Downcasting avec “as?”
- Types spéciaux: Any (N’importe quoi), AnyObject (N’importe quelle instance)

# Extensions

- Ajouter des propriétés
- Méthodes d'instance
- Nouveaux initialiseurs
- Subscripts
- Définir de nouveaux types imbriqués
- Se conformer à un protocole

```
extension SomeType : SomeProtocol , AnotherProtocol {  
    // implementation of protocol requirements goes here  
}
```



# Protocols

- Définissent des méthodes / propriétés et autres pré-requis pour une tâche particulière ou une fonctionnalité (= Contrat)
- Les protocoles sont des types (utilisation possible de “is”/“as” pour vérifier qu’on est conforme à un protocole)
- Utilisés pour l’utilisation de la délégation (pattern récurrent de Cocoa)

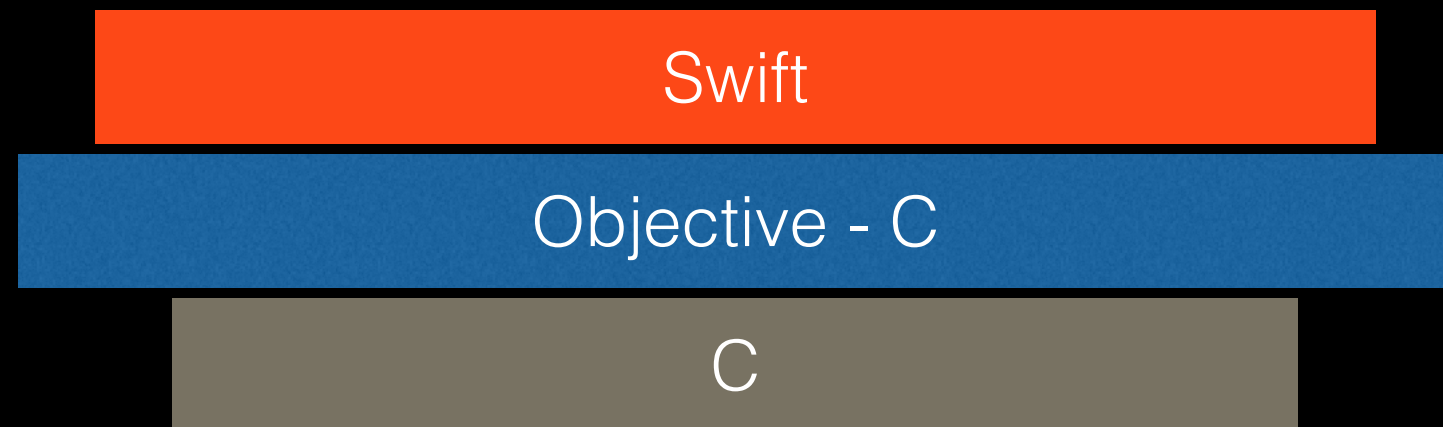
```
protocol SomeProtocol {  
    // protocol definition goes here  
}
```

# More things...

- Generics
- Surcharge d'opérateur
- Enchaînement d'optionnels
- Types Imbriqués
- Capture (unowned)
- Subscripts ...
- Labelled Statements

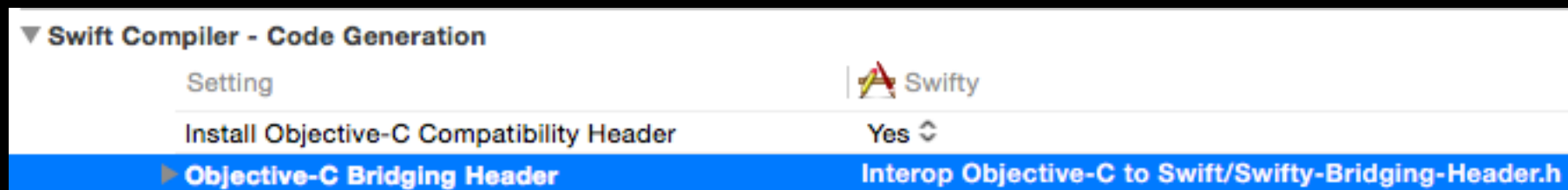
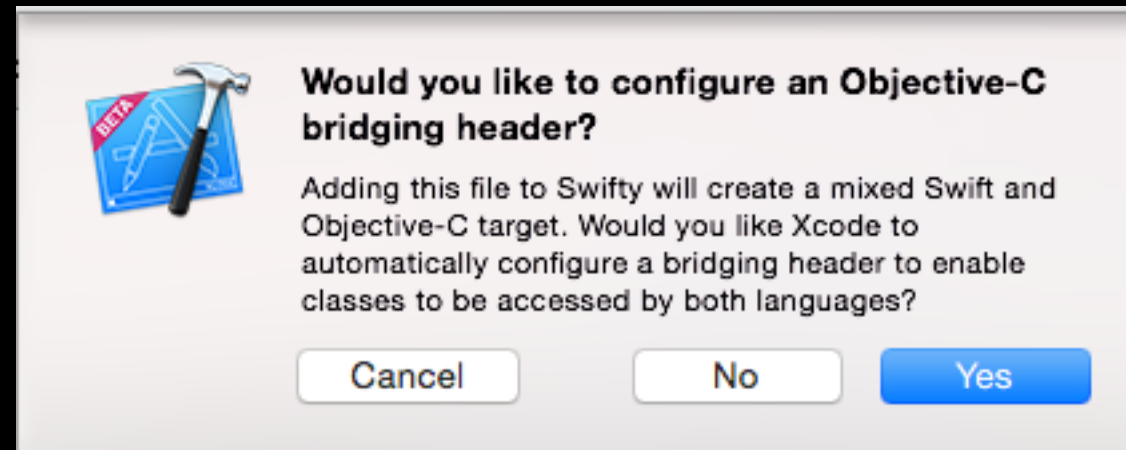
# Compatibilité

- Xcode 6 (Actuellement en B1)
- iOS 7+ / OS X 10.9+



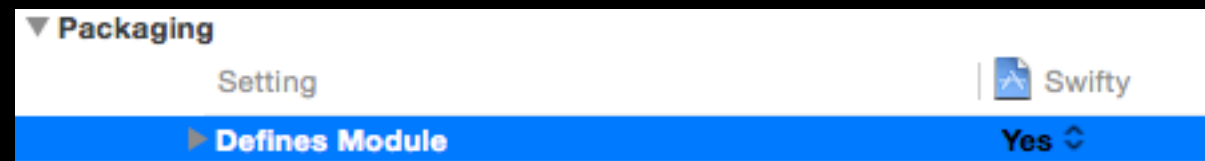
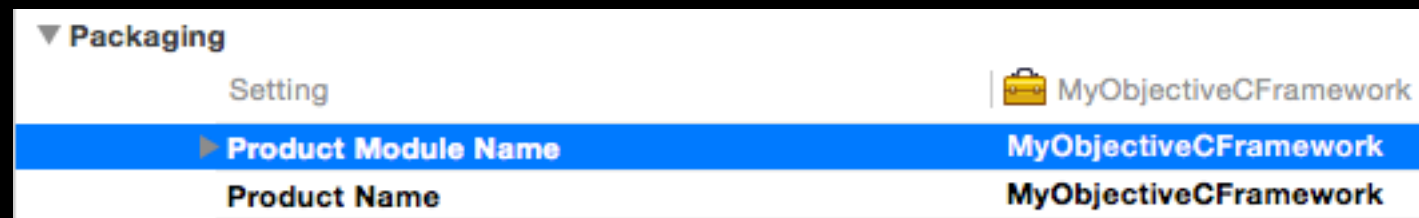
# Interopérabilité

- Demo - Objective-C vers Swift



# Interopérabilité

- Demo - Swift vers Objective-C



# Migration vers Swift ?

- Pas nécessaire, Objective-C reste maintenu
- Swift est une autre possibilité pour le dev iOS/OS X
- Grande interopérabilité entre les 2 languages
- Toujours en Beta - risque d'évoluer

# Ressources



- Sessions WWDC14
- Devforums Apple
- @Florian\_MrCloud sur twitter
- <https://github.com/MrCloud/Swifty>

# Play with Swift

- Playground - Demo