

Développement iOS

Swift 2.0 Basics

Utilisation de Swift 2.0 sans POO



Sommaire

1. Généralités
2. « PlayGround » et « Story Board »
3. Variables et constantes (Structures de stockage)
4. Opérateurs et support UNICODE (EMOJI)
5. Conditions, boucles (Structures de contrôle)
6. Tableaux, dictionnaires
7. Fonctions
8. Balises



Histoire

- WWDC 2014, 2 juin 2014
- Swift nouveau langage pour les Frameworks Cocoa et Cocoa Touch
- Swift pour OSX et iOS
- Nécessite Xcode >= 6.xx.yy La documentation de base 1000 pages (iBooks)
- Xcode 7.0.0 autorise l'installation d'une App sans le programme de développement standard.
- Ancienneté : 1 an et demi (très jeune)
- Peu de sources en français !
- Ce livre pour iOS 8 : <http://livre.fnac.com/a7793093/T-Sarlandie-Programmation-mobile-IOS-8-avec-Swift>
- Nous sommes déjà à iOS 9 !
- Les ressources principalement en anglais ont le mérite d'être à jour !



Environnement

- Swift se met en oeuvre :
 - Dans des projets : File -> new - > project
 - Dans le terrain de jeux : File -> new -> Playground
- Playground permet de tester nativement sans compilation du code de manière plutôt visuelle (débutant)
- Playground semble ne pas être encore totalement stable, au fur et à mesure de l'allongement du code (même avec Xcode 7.0.0)
- Playground est un bundle, comme toutes les autres Apps.



PlayGround



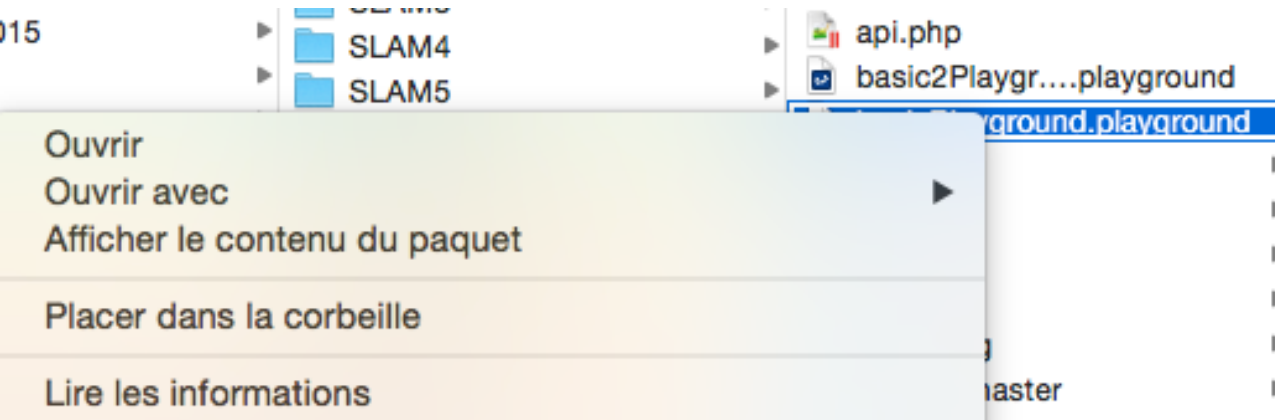
- Code and Play !
- PlayGround est un dispositif relativement particulier, avec une technologie spécifique.
- PlayGrounds enable the interactive experience of a script language.
- But you can see some trouble, with complex code.
- PlayGround interprète du code en temps réel à chaque modification.



PlayGround

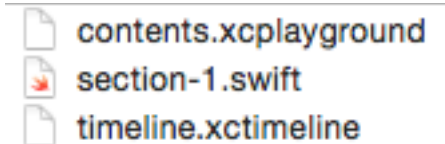


- Un fichier playGround est en fait un bundle



basicPlayground.playground

- Le paquet contient les éléments suivants :
- On peut trouver un dossier Ressources et Documentation



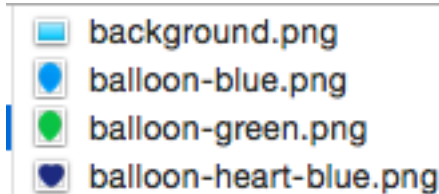
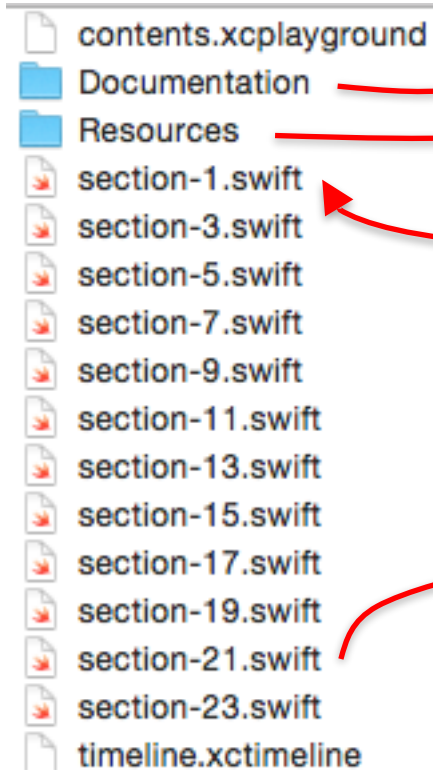
+ PlayGround

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<playground version='3.0' sdk='macosx'>
  <sections>
    <code source-file-name='section-1.swift' />
    <documentation relative-path='fragment0.html' />
    <code source-file-name='section-3.swift' />
    <documentation relative-path='fragment11.html' />
  </sections>
</playground>
```



- Prenons le cas du projet Balloon :



HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>TITLE</title>
    <link rel="stylesheet" type="text/css" href="style-1.1.15.css" />
    <meta charset="utf-8">
    <meta id="xcode-display" name="xcode-display" content="render" />
    <meta name="apple-mobile-web-app-capable" content="yes" />
    <meta name="viewport" content="width=device-width, maximum-scale=1.0">
  </head>
  <body id="conceptual_flow_with_tasks" class="jazz">
    <div class="content-wrapper">
      <article class="chapters">
        <section class="section">
          <p class="para">
            This playground includes sample code to explore SpriteKit. Th
            was shown on stage at WWDC 2014, contains two cannons tha
```

Swift

```
let sceneView = SKView(frame: CGRect(x: 0, y: 0, width: 850, height: 638))
let scene = SKScene(fileName: "GameScene")
scene.scaleMode = .AspectFill
sceneView.presentScene(scene)

XCPShowView("Balloons", sceneView)
```

+ PlayGround

- Avec la configuration du bundle, vous proposez des exemples de codes commentés :
- Avec du HTML, et donc des liens hypertextes, et des animations, ...
- Le projet Ballon réalisé par l'équipe Swift est intéressant à examiner.





Story Board

- Le « Story Board » permet de concevoir l'interface d'une application iOS.
- Pour les applications « console » dans OSX, le « Story Board » n'est pas utilisé.
- Les exemples proposés dans ce support ne permettent pas de mettre en oeuvre le « Story board ».



Basics

- Plus de points virgules (; semi colon) à la fin des instructions
- Toujours des { (alt+5) et des } (alt+°) pour encadrer des blocs
- Swift gère l'inférence, c'est à dire qu'il n'est pas nécessaire de mettre un type pour déclarer une variable
- « PlayGround » permet de s'initier simplement aux Basics
- Les exemples qui suivent sont basés sur Swift 2.0 (il peut rester quelques instructions en Swift 1.2)
- Println est remplacé par print en Swift 2.0

+ Variables et constantes

- Var = variables pour tous les types (mutable)
- Let = constante pour tous les types (not mutable)

```
// variable avec inférence
var str = "Hello, playground"
// variable typée
var valeur : Int = 9
//constante inférente
let a = valeur
//constante typée
let constante : Float = 3.14159265353
```

```
"Hello, playground"
9
9
3.14159274101257
```

- Print() permet d'afficher des messages dans la console !

```
// Chaînes
var maChaine : String = "Bonjour"
println(maChaine)
println("mon message : \(maChaine)")
maChaine = maChaine + " le monde" + " est beau"

var designers = 1
var developers = 4
let myTeamProject = "\(designers) Designers \
(developers) Developers = \(designers +
developers) "
println(myTeamProject)
```

```
"Bonjour"
"Bonjour"
"mon message : Bonjour"
"Bonjour le monde est beau"

1
4
"1 Designers 4 Developers = 5 "
"1 Designers 4 Developers = 5 "
```

+ Variables et constantes

- Pensez aux constructeurs :

```
//Initializing Strings
var uneChaine = ""
var uneAutreChaine = String()
uneAutreChaine = "C'est bon"
```

```
""
""
"C'est bon"
```

- Attention les « Character » prennent des " "

```
//Characters
var unC : Character = "$"
uneAutreChaine.append(unC)
|
```

```
"$"
"C'est bon$"
```

var	variable		assign initial
keyword	name	type	value
<hr/>			
var	runningTotal	: Double	= 0.0

var	variable	assign initial
keyword	name	value
<hr/>		
var	runningTotal	= 0.0

+ Variables et constantes

- Conversions de types entre Int et String :
- String to Int (cast) :

```
// Int et String
var nbInt : Int = 0
var ch : String = "111111"

// String to Int
let nb = Int(ch)
print("valeur de nb : \((nb!+1)")
```

```
0
"111111"

111 111
"valeur de nb : 111112\n"
```

- Int to String (cast) :

```
nbInt = 123456

// Int to String
ch = String(nbInt)
ch = ch + "\n" + "\n" + "\n"
print("valeur de ch :")
print(ch.stringByTrimmingCharactersInSet(NSCharacterSet.whitespaceAndNewlineCharacterSet()))
```

```
123 456

"123456"
"123456\n\n\n"
"valeur de ch :\n"
"123456\n"
```

+ Les t-uples

- Un t-uple est une variable qui peut contenir plusieurs valeurs.
- Chaque valeur peut être reliée à une étiquette, comme dans un tableau associatif :

```
//t-uples
```

```
let record = (cle : "1", valeur : "abc")  
let key = record.cle  
let value = record.valeur
```

```
(.0 "1", .1 "abc")  
"1"  
"abc"
```

+ Les opérateurs

- +, -, *, /, %
- Modulo : %

```
// Modulo
```

```
let division = 100 / 11 // Dans 100 combien de fois 11 ?  
let modulo = 100 % 11 // Quel est le reste de 100 / 11 ?
```

```
/*  
Dans 100 il y a 9 fois 11  
Il reste alors 1  
*/  
print(division) // Affiche 9  
print(modulo) // Affiche 1
```

```
9  
1
```

```
"9\n"  
"1\n"
```

+ Support UNICODE

- Strings in Swift are fully Unicode compliant, so you can use EMOJI characters inside string.

- Les noms de variables peuvent comporter des caractères UNICODE :

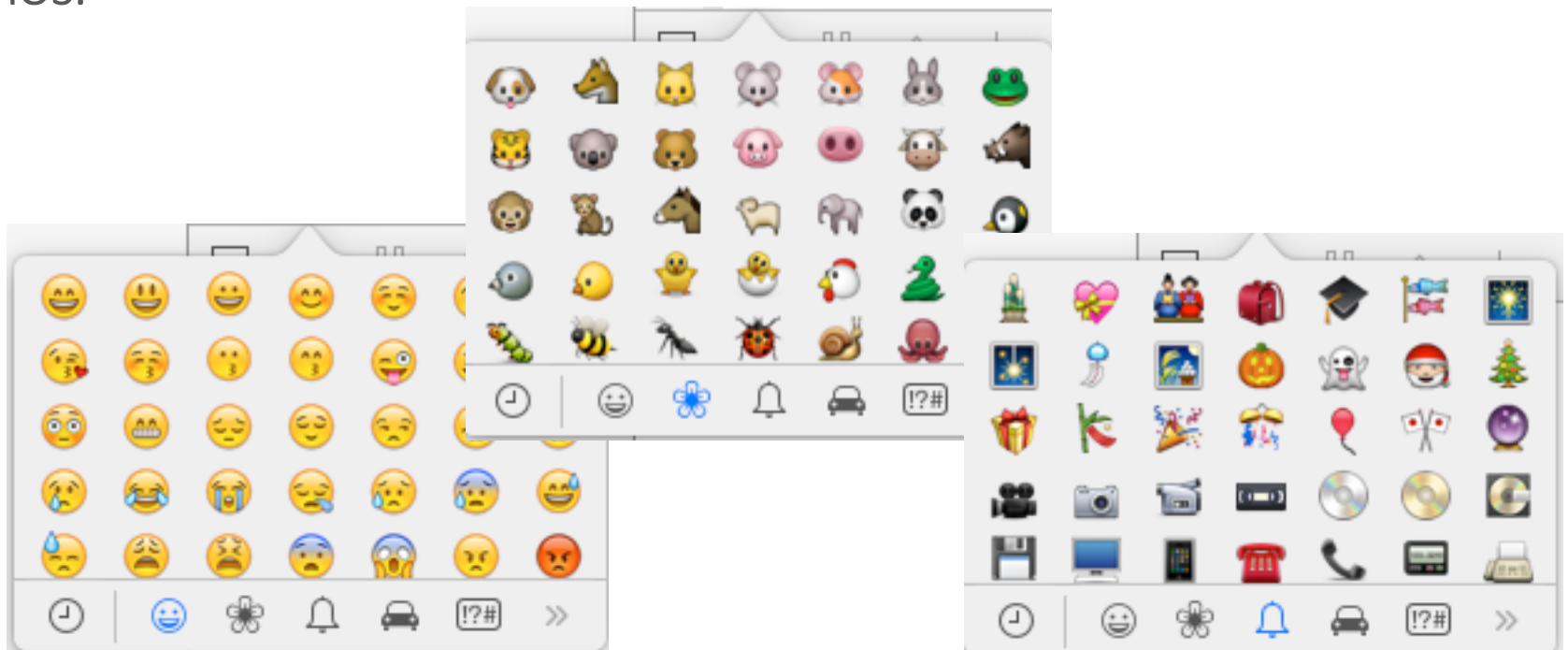
```
var π = 3  
println(π)
```

- Vous pouvez mettre des EMOJI dans vos chaînes :

```
let multipleHearts = "❤️❤️❤️❤️"
```


+ Support UNICODE

- Comment avoir accès aux caractères EMOJI dans OSX ?
- Ctrl + Cmd + espace : permet d'avoir accès aux EMOJI
- Les EMOJI ne sont supportés que dans les applications natives OSX et iOS.





Conditions

- La structure alternative If :

```
//Conditions
```

```
let age = 12
```

```
if age >= 18 {  
    print("Vous êtes majeur !")  
} else {  
    print("Vous êtes mineur.")  
}
```

12

"Vous êtes mineur.\n"

```
let age2: Int = 32
```

```
let nationalite: String = "USA"
```

```
if age2 >= 21 && nationalite == "USA" {  
    print("Vous êtes Américain et en plus vous êtes majeur.")  
}
```

32

"USA"

"Vous êtes Américain et en plus vous êtes majeur.\n"



Conditions

- Opérateurs booléens à utiliser dans les conditions :

Signe	Signification	Vrai quand ?
&&	et	Vrai quand les deux valeurs sont vraies.
	ou	Vrai quand au moins une des deux valeurs est vraie.

+ Conditions

- Opérateurs à utiliser dans les conditions :

Signification	Opérateur
Parenthèses	(,)
Non	!
Multiplication, division, modulo	*, /, %
Addition soustraction	+, -
Inférieur, inférieur ou égal, supérieur, supérieur ou égal	<, <=, >, >=
Égal, différent de	==, !=
Et	&&
Ou	

- La structure Switch :

```
let note = 18

switch note {
case 10:
    print("Vous avez la moyenne, mais vous n'obtenez pas de mentions.")

case 12:
    print("Vous avez obtenu la mention assez bien.")

case 14:
    print("Vous avez obtenu la mention bien.")

case 16:
    print("Vous avez obtenu la mention très bien.")

case 18:
    print("Vous avez les félicitations du jury !")

default:
    print("Navré, il faut avoir une de ces notes pour avoir une mention.")
}
```

"Vous avez les félicitations du jury !\n"

+ Conditions

- Dans le cas de conditions spéciales avec Switch :

```
switch note {  
case 0,1,2,3,4,5,6,7,8,9:  
    print("Vous n'avez pas la moyenne, vous n'avez donc pas de mention.")  
  
case 10, 11:  
    print("Vous avez la moyenne, mais vous n'obtenez pas de mentions.")
```

```
switch note {  
case 0...9:  
    print("Vous n'avez pas la moyenne, vous n'avez donc pas de mention.")  
  
case 10...12:  
    print("Vous avez la moyenne, mais vous n'obtenez pas de mentions.")  
  
switch note {  
case 0..<10:  
    print("Vous n'avez pas la moyenne, vous n'avez donc pas de mention.")  
  
case 10..<12:  
    print("Vous avez la moyenne, mais vous n'obtenez pas de mentions.")
```

+ Ternaires

- Les conditions ternaires sont utilisables à la place de If :

```
let notes = 12
var moyenne: Bool

// Cette partie
if notes < 10 {
    moyenne = false
} else {
    moyenne = true
}

// Est identique à cette partie
moyenne = notes < 10 ? false : true
```

12

true

true

- While :

```
var nbDeLignes: Int = 1

while nbDeLignes <= 1000 {
    print("Je dois apprendre mes leçons en cours de Swift.")
    nbDeLignes++ // Equivalent à : nbDeLignes = nbDeLignes + 1
}
```

(1000 times)
(1000 times)

- Repeat ... While

```
repeat {
    print("\(nbDeLignes). Je dois apprendre mes leçons en cours de Swift.")
    nbDeLignes++
} while nbDeLignes <= 1000
```

"1001. Je dois apprendre mes leçons en cours de Swift.\n"
1 001

- For, in

```
for nbDeLignes in 1...1000 {
    print("\(nbDeLignes). Je dois apprendre mes leçons en cours de Swift.")
}
```

(1000 times)

+ Boucles

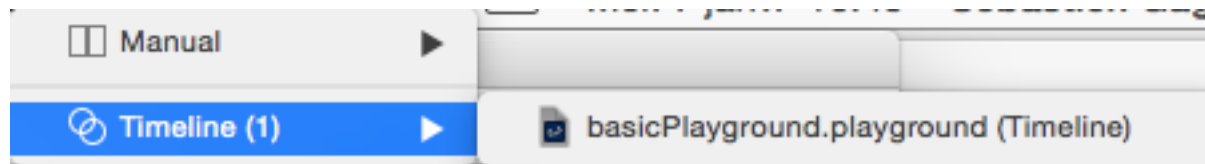
- For avec pas de deux :

```
// pas de deux  
  
var multiplesDeDeux: Int  
  
for multiplesDeDeux = 0; multiplesDeDeux <= 100; multiplesDeDeux = multiplesDeDeux + 2 {  
    print(multiplesDeDeux)  
}
```

(51 times)

+ Tableaux (Arrays)

- PlayGround autorise la TimeLine afin d'avoir une vue console



- Un exemple simple :

```
//Arrays
var maTable = [42, "arbre", "Pomme", "newton", "apple", 2]
print(maTable[1])
print(maTable[0])
for item in maTable {
    print(item)
}
// Add another item in array
maTable.insert("ee", atIndex: 0)
for item in maTable {
    print(item)
}

//Arrays
var tab : Array = ["a", "b", "c", 3, 4, 5 ]
```

```
[42, "arbre", "Pomme", "newton", "apple", 2]
"arbre\n"
"42\n"

(6 times)

["ee", 42, "arbre", "Pomme", "newton", "apple", 2]

(7 times)

["a", "b", "c", 3, 4, 5]
```

+ Tableaux (Arrays)

- Il est possible d'initialiser un tableau en prévision des futures valeurs prises :

```
var animalArray = [String]()  
animalArray = ["cow", "dog", "cat", "mouse"]  
print(animalArray.description)  
  
var animalAgeArray = [Int]()  
animalAgeArray = [1, 1, 2, 3]  
print(animalAgeArray.description)  
if (animalArray.isEmpty)  
{  
    print("vide")  
}  
else  
{  
    print("non vide")  
}
```

```
[]  
["cow", "dog", "cat", "mouse"]  
["cow", "dog", "cat", "mouse"]\n  
  
[]  
[1, 1, 2, 3]  
[1, 1, 2, 3]\n  
  
"non vide\n"
```

- Les tableaux sont toujours « mutables »

+ Tableaux (Arrays)

- Une autre méthode pour déclarer un tableau vide :

```
//Arrays Always
// Empty Arrays
var tabX:[String] = []
//inferred syntax
var colorArray:Array<String> = ["Red","Blue","Green"]
var colorArray1:[String] = ["Red","Blue","Green"]
var colorArray2 = ["Red","Blue","Green"]
```

```
[]
["Red", "Blue", "Green"]
["Red", "Blue", "Green"]
["Red", "Blue", "Green"]
```

- Modifier des valeurs :

```
colorArray[0] = "Brown"
colorArray[1...2] = ["blue","green"]
```

```
"Brown"
["blue", "green"]
```

- Supprimer des éléments :

```
//Remove item
colorArray.removeAtIndex(1)
colorArray.removeLast
colorArray.removeAll()
```

```
"blue"
"green"
[]
```

+ Tableaux (Arrays)

- Ajouter des éléments :

```
//Add item

colorArray1.insert("Red",atIndex:1)
colorArray2 += colorArray1
colorArray2 = colorArray2.reverse()
print(colorArray2)
```

```
["Red", "Red", "Blue", "Green"]
["Red", "Blue", "Green", "Red", "Red", "Blue", "Green"]
["Green", "Blue", "Red", "Red", "Green", "Blue", "Red"]
["Green", "Blue", "Red", "Red", "Green", "Blue", "Red"]\n"
```

- La fonction « reverse » permet d'inverser un tableau
- Tableau d'entiers initialisé avec un constructeur :

```
// Array 2.0

var monTab : [Int] = [Int]()
monTab[0] = 1
```

```
[]
1
```

+ Dictionaries

- Quelques instructions de base :

```
//Dictionaries
// keyes values pairs
var monDict = ["France":"Paris","USA":"New York",
               "Allemagne": "Berlin"]
print("Nombre de couples : \(monDict.count)")
print("description : \(monDict.description)")
monDict.updateValue("San Fransisco", forKey: "USA")
print("description : \(monDict.description)")
monDict.removeValueForKey("France")
for (country, town) in monDict {
    print("Key-> \(country) Value-> \(town)")
}
```

```
["Allemagne": "Berlin", "France": "Paris", "USA": "New York"]
"Nombre de couples : 3\n"
"description : ["Allemagne": "Berlin", "France": "Paris", "USA":...
"New York"
"description : ["Allemagne": "Berlin", "France": "Paris", "USA":...
"Paris"
(2 times)
```

- Lien avec NSDictionary :

```
// Old style NSDictionary
var unDict : NSMutableDictionary
unDict = NSMutableDictionary()

//Only available with Mutable Dictionary
unDict.setValue("valeur1", forKey: "clé1")
unDict.setValue("valeur1", forKey: "clé2")
unDict.setValue("valeur1", forKey: "clé3")
```

```
["clé1": "valeur1"]
["clé2": "valeur1", "clé1": "valeur1"]
["clé2": "valeur1", "clé1": "valeur1", "clé3": "valeur1"]
```



Dictionaries

- Empty, inferred, explicit, ..., déclarations :

```
//Empty dictionary
var emptyDictionary1 = Dictionary<String, String>()
// or
var emptyDictionary2 = [String: String]()
//inferred
var personAge1 = ["Larry King": 43, "Mike Johnson":32, "Ted
    Brown": 67]
var personAge2 = ["Larry King": 43, "Mike Johnson":32, "Ted
    Brown": 67]

// Explicit
var personAgeExplicit1:Dictionary<String, Int> = ["Larry King":
    43, "Mike Johnson":32, "Ted Brown": 67]
// or
var personAgeExplicit2:[String: Int] = ["Larry King": 43, "Mike
    Johnson":32, "Ted Brown": 67]
```

- Dictionaries are mutable by default

```
[:]
```

```
[:]
```

```
["Mike Johnson": 32, "Larry King": 43, "Ted Brown": 67]
["Mike Johnson": 32, "Larry King": 43, "Ted Brown": 67]
```

```
["Mike Johnson": 32, "Larry King": 43, "Ted Brown": 67]
```

```
["Mike Johnson": 32, "Larry King": 43, "Ted Brown": 67]
```



Parcours de tableaux ou dictionnaires

- For :

```
// parcours de tableaux ou dictionnaires
```

```
let prenom = ["Camille", "Maxime", "Antoine", "Lucie", "Mathilde"]  
var i: Int
```

```
print("Les membres de la famille sont :")  
for i = 0; i < 5; i++ {  
    print("- " + prenom[i])  
}
```

```
["Camille", "Maxime", "Antoine", "Lucie", "Mathilde"]
```

```
"Les membres de la famille sont :\n"
```

```
(5 times)
```

- For in :

```
for prenom in prenom {  
    print(prenom)  
}
```

```
(5 times)
```


+ Parcours de tableaux ou dictionnaires

- For in avec une clé :

```
let personne = ["Nom": "Durand", "Prénom": "Maxime", "Adresse": "94 rue machin", "Ville": "Lille"]  
for (cle, valeur) in personne {  
    print(cle + " - " + valeur)  
}
```

```
["Nom": "Durand", "Ville": "Lille", "Prénom": "Maxime", "Adress...
```

```
(4 times)
```

+ Fonctions

- Possibilité de retourner plusieurs valeurs :

```
func addition (val1 : Int, val2 : Int) -> Int {  
    return val1+val2  
}
```

```
addition(3,5)
```

```
func calcul (val1 : Float, val2 : Float, val3 :  
    Int ) -> (add :Float, div : Float)  
{  
    return(val1+val2, val1/val2)  
}
```

```
let addition = calcul(3,5,6).add  
let divition = calcul(5,6,7).div
```

8

8

(2 times)

8.0

0.8333333313465118

+ Fonctions

- Une fonction comme paramètre :

```
func premierPlusGrandQueDeuxieme(nb1: Int, nb2: Int) -> Bool {  
    return nb1 > nb2 // nb1 > nb2 retourne un booléen si c'est vrai ou non  
}
```

```
func maFonction(maFonctionParametre: (Int, Int) -> Bool) {  
    if maFonctionParametre(4, 3) {  
        print("Condition validée.")  
    }  
}
```

```
// On a juste à fournir le nom de notre fonction  
// A condition qu'elle respecte les paramètres et type de retour  
maFonction(premierPlusGrandQueDeuxieme)  
  
// Affichera "Condition validée"
```

true

"Condition validée.\n"

+ Fonctions

- Une fonction comme type de retour :

```
func hello(debutMessage: String) -> (String) -> String {  
    func nestedHello(finMessage: String) -> String {  
        return "\(debutMessage) \(finMessage)"  
    }  
    return nestedHello  
}  
  
print(hello("Hello, ")(("World !"))  
// Affichera "Hello World !"
```

"Hello, World !"

String -> String

"Hello, World !\n"

+ Fonctions

- Les « closures » sont des fonctions sans nom :

```
func maFonction2(maFonctionParametre: (Int, Int) -> Bool) {  
    if maFonctionParametre(4, 3) {  
        print("Condition validée.")  
    }  
}  
  
maFonction2({ (nb1: Int, nb2: Int) -> Bool in  
    return nb1 > nb2  
})  
  
// Affichera "Condition validée"
```

"Condition validée.\n"

true

- Cela permet de simplifier le code, mais la compréhension est moins évidente !

+ Organize your code

- What kind of developer are you ?

ORGANIZE YOUR CODE
MARK TODO FIXME

- Organize your Swift code with the newest additions, MARK, TODO & FIXME.
- MARK : `//# MARK:` - Text Methods
- TODO : `//# TODO:` - Add social features
- FIXME : `// FIXME:` - Bug in 2.0

+ Organize your code

- Example :

```

    }
    return dict!
}

//# MARK: - Merge Dictionaries
func mergeDictionaries(firstDict : NSMutableDictionary,
```

- Where find organized code :

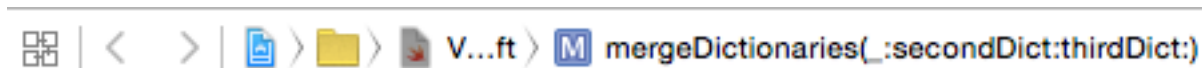


Diagram illustrating the breadcrumb navigation bar in Xcode, showing the path: `M mergeDictionaries(_:secondDict:thirdDict:)`.



Diagram illustrating the breadcrumb navigation bar in Xcode, showing the path: `M mergeDictionaries(_:secondDict:thirdDict:)`.