Organisation du module

camille.guinaudeau@u-psud.fr

Organisation du module

Planning du semestre :

- S8 : CM1
- S10 : TP1
- S11 : TP2
- S13 : TP3 + TP non encadré 1
- S14 : CM2 + TP4
- S15 : TP 5
- S19 : TP non encadré 2 + TP6
- S20 : TP noté
- S21 : TP non encadré 3 + TP7
- S22 - 24 : Projet

Evaluation :

- **CC :** TP noté + projet (40%)
- **DS :** Sur papier, tous documents autorisés (60%) - fin mars / début avril

Xcode et Langage Swift

camille.guinaudeau@u-psud.fr

Programmation iOS

Pourquoi pas Cordova ?

L'objectif du cours est de vous apprendre à utiliser les outils d'Apple dédiés à la programmation iOS

Cordova :

- framework open-source
- permet de créer des applications pour différentes plateformes (Android, iOS, Ubuntu, Windows 8…)
- HTML, CSS et JavaScript

—> Nouveau langage, manque de développeurs « Swift »

Programmation iOS

- La programmation sous iOS fait intervenir deux éléments :
  - votre code en Swift
  - l'API Cocoa Touch

—> Programmation orientée-objet

- La création de nouvelles applications se fait par le biais de l'IDE Xcode

Plan du cours

1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

Plan du cours

1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

Swift

Swift est le nouveau langage de programmation pour iOS et OSX lancé par Apple en juin 2014

- plus concis que Objective-C
- compatible avec Objective-C

Nouvelles versions du langage lancées régulièrement
—> Actuellement Swift 4
—> Modification de la syntaxe

Cocoa Touch

Le layer Cocoa Touch contient les frameworks pour la création d'applications iOS

Ces frameworks permettent de :

- définir l'apparence de votre application,
- fournir l'infrastructure de base des applications,
- fournir la prise en charge des technologies clés: le multitâche, les entrées tactiles, reconnaissance de gestes, etc.

Exemple de frameworks utilisés dans ce module:

- Foundation Kit
- UIKit
- Message UI

Programmation réactive

Aucune partie de votre code ne va s'effectuer tant qu'elle n'a pas été appelée par Cocoa Touch

Programmer sous iOS consiste à savoir quand et pourquoi Cocoa Touch appelle votre code
—> Votre code va se lancer en réponse à un **évènement**

Un événement peut être :

- lié aux actions de l'utilisateur
  - appuyer sur un bouton
  - entrer une valeur dans un champ de texte
- lié au cycle de vie de l'application
  - chargement de l'écran d'accueil de l'application
  - passage en arrière plan

Plan du cours

1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

Plan du cours

1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

IDE Xcode

Xcode est l'environnement de développement pour Mac OS X ainsi que iOS

Les langages

C, C++, Objective-C, Java, AppleScript, Python, Ruby, Rez et Swift ainsi que les API Carbon et Cocoa

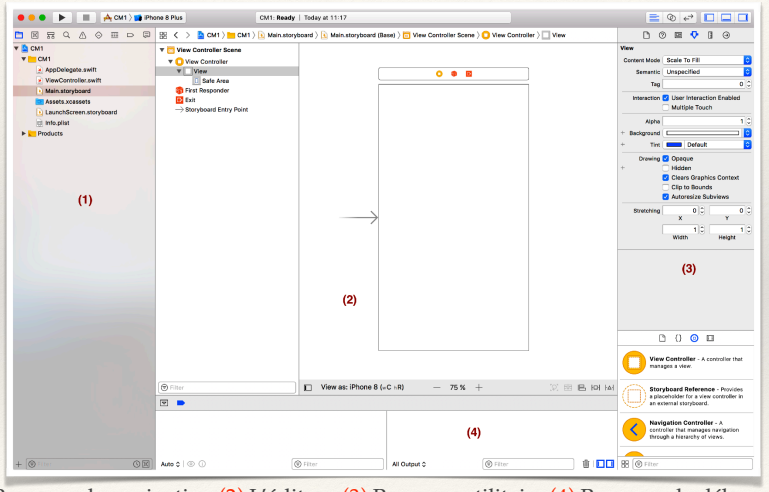
Interface Builder

—> Outils spécifique à Xcode qui permet de construire les interfaces graphiques

Xcode contient également la documentation d'Apple

IDE Xcode

La fenêtre « Projet »

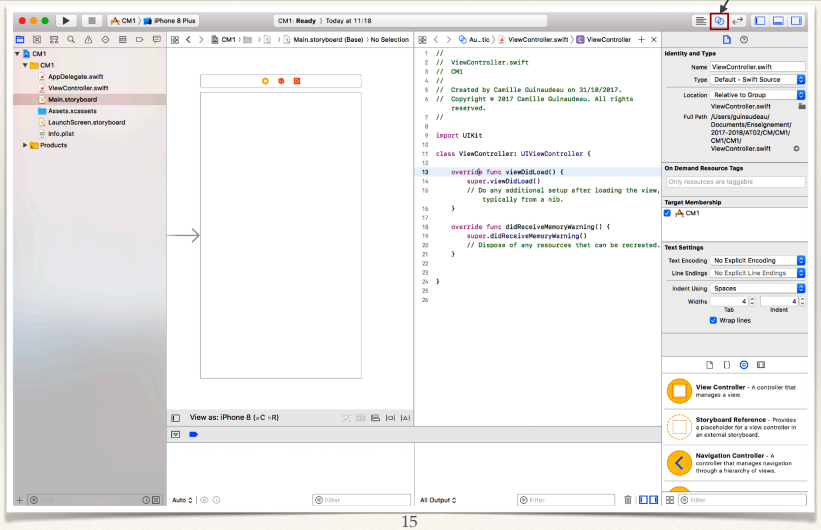


(1) Panneau de navigation (2) L'éditeur (3) Panneau utilitaire (4) Panneau de débuggage

IDE Xcode

L'assistant d'édition

Bouton permettant d'activer l'assistant d'édition



IDE Xcode

Comprendre le code généré par Xcode

Lors de la création du projet, Xcode produit plusieurs fichiers :

- AppDelegate.swift : fichiers de définition et de code de la classe AppDelegate qui gère l'ensemble des évènements de la classe application (démarrage, fin de chargement, etc.)
- ViewController.swift : fichiers de définition et de code de la classe ViewController qui définit le contrôleur de la vue de l'application
- Main.storyboard : fichier qui va contenir graphiquement tous les éléments qui seront présentés à l'utilisateur
- Assets.xcassets et LauchScreen.storyboard : fichiers qui vont permettre de définir l'icône et l'écran de lancement de l'application
- Info.plist : fichier de propriétés qui fournit des informations sur l'application (quel est son nom ? quelle icône faut-il afficher ? etc.)

Classe AppDelegate

Délégué de l'application

AppDelegate.swift

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool

func applicationWillResignActive(_ application: UIApplication)

func applicationDidEnterBackground(_ application: UIApplication)
func applicationWillEnterForeground(_ application: UIApplication)
func applicationDidBecomeActive(_ application: UIApplication)
func applicationWillTerminate(_ application: UIApplication)
```


Classe ViewControlleur

Contrôleur de vue

Appareils iOS : une seule fenêtre sur l'écran

Interface : vues empilées qui seront affichées selon les directives du programmeur

Chaque vue est constituée d'un ou de plusieurs éléments :

- informations (textes, images, vidéos)
- éléments de communication avec l'utilisateur (boutons, zones de texte, curseurs, etc.)
- éléments de navigation entre les différentes vues (onglets, barres de navigation, barres de recherche, etc.)

Chaque vue est accompagnée par son **ViewController** qui est responsable de la gestion de la vue et de toutes les actions qu'elle peut générer (chargement de la vue, réaction aux actions d'un utilisateur par exemple, gestion de la mémoire)

Classe ViewControlleur

Contrôleur de vue

Lors de la création d'un nouveau contrôleur de vue, Xcode fournit la définition de deux méthodes

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a
nib.
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}
```

ViewController.swift

Interface Builder

Rôle de l'Interface Builder (IB)

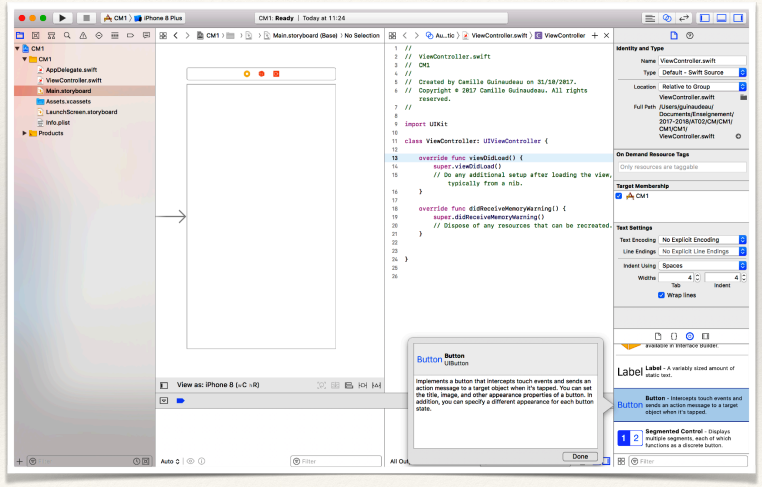
Interface Builder
Outils de développement d'interface graphique pour des applications OS X et iOS

Programmation par « glisser-déposer »
Permet de créer facilement une interface sans avoir à écrire du code

Les interfaces graphiques générées grâce à Interface Builder sont archivées dans un fichier nib (NextStep Interface Builder).

Interface Builder

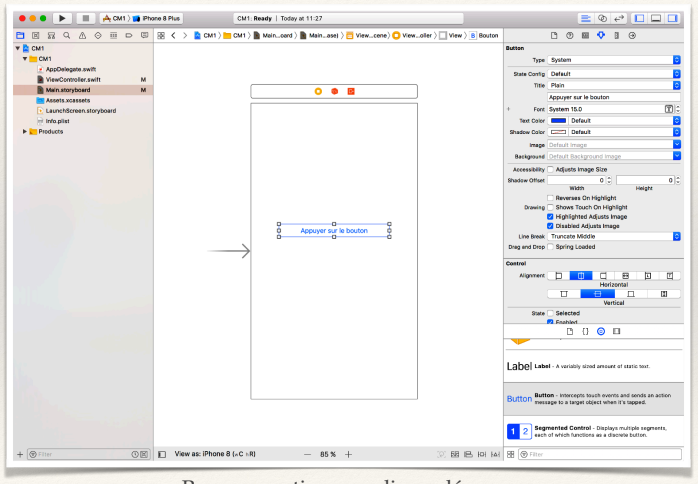
Rôle du Storyboard



Programmation par glisser-déposer

Interface Builder

Rôle du Storyboard



Programmation par glisser-déposer

Main.Storyboard

Lien entre l'interface et le code

IBOutlet

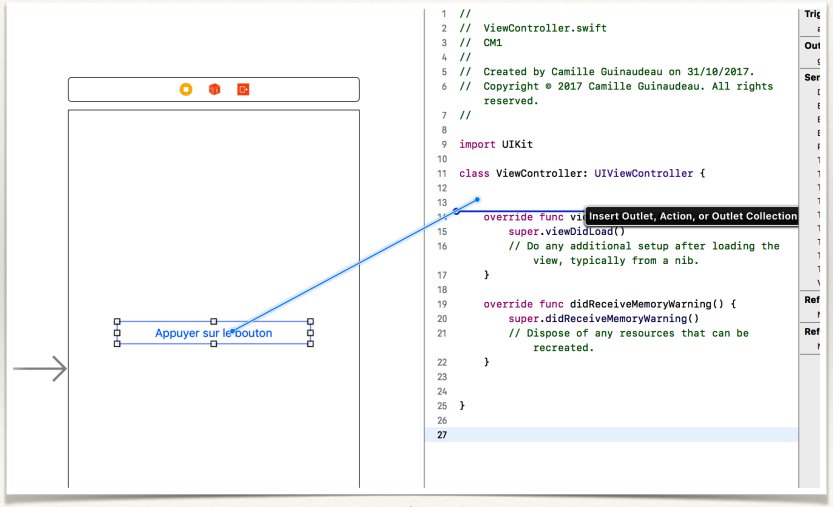
- variable d'instance de la classe qui contient un référence vers un objet
- créé une connexion entre le code et l'objet dans le storyboard

IBAction

- remplace le type de retour void
- permet de signifier à l'interface Builder que la méthode correspond à une action qui doit être effectuée en réponse à un évènement

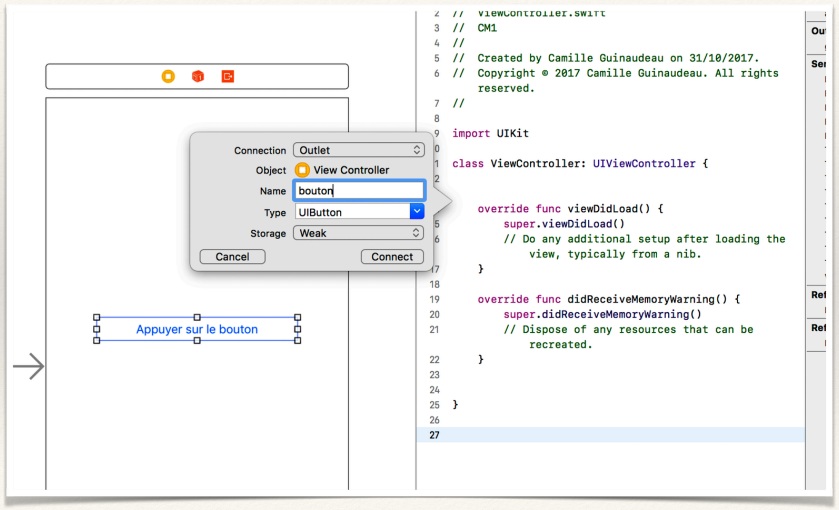
Main.Storyboard

Lien entre l'interface et le code



Main.Storyboard

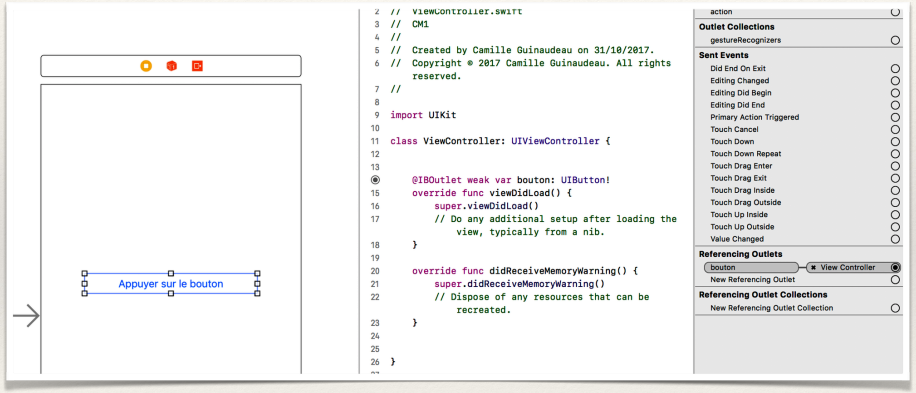
Lien entre l'interface et le code - IBOutlet




```swift
//  ViewController.swift
//  CM1
//
//  Created by Camille Guinaudeau on 31/10/2017.
//  Copyright © 2017 Camille Guinaudeau. All rights
    reserved.
//

import UIKit

class ViewController: UIViewController {


    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the
            view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be
            recreated.
    }


}
```


Main.Storyboard

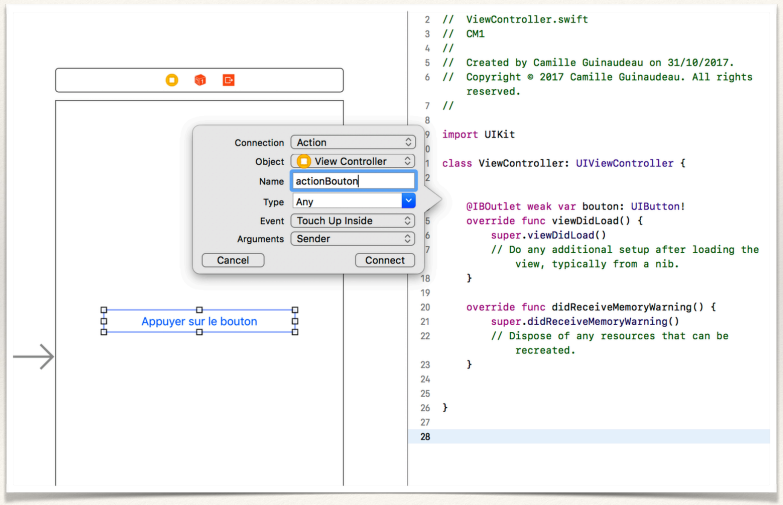
Lien entre l'interface et le code - IBOutlet




```swift
//  ViewController.swift
//  CM1
//
//  Created by Camille Guinaudeau on 31/10/2017.
//  Copyright © 2017 Camille Guinaudeau. All rights
    reserved.
//

import UIKit

class ViewController: UIViewController {


    @IBOutlet weak var bouton: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the
            view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be
            recreated.
    }


}
```


Main.Storyboard

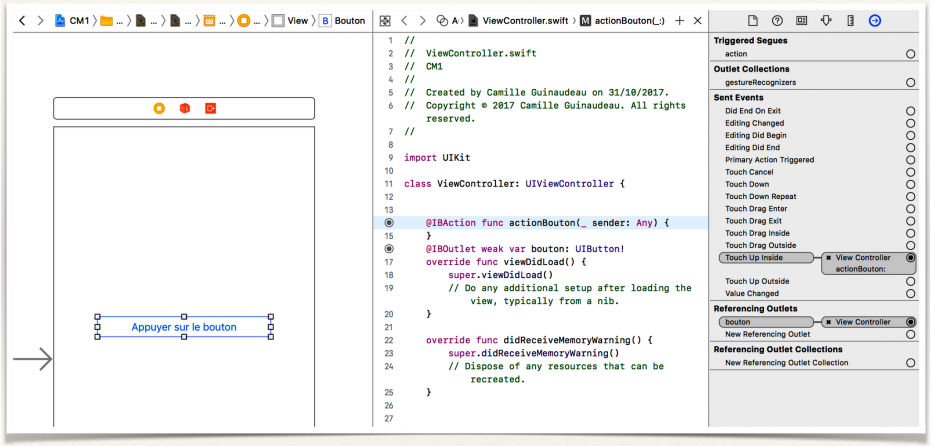
Lien entre l'interface et le code - IBAction




```swift
//  ViewController.swift
//  CM1
//
//  Created by Camille Guinaudeau on 31/10/2017.
//  Copyright © 2017 Camille Guinaudeau. All rights
    reserved.
//

import UIKit

class ViewController: UIViewController {


    @IBOutlet weak var bouton: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the
            view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be
            recreated.
    }


}
```


Main.Storyboard

Lien entre l'interface et le code - IBAction



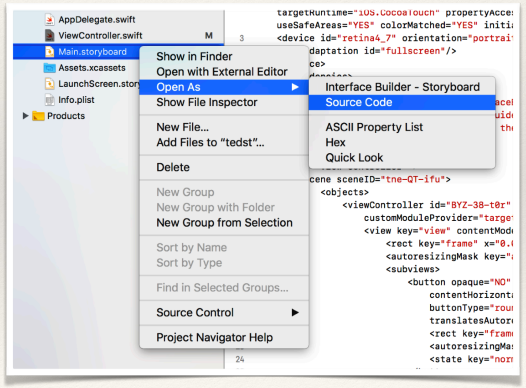

```swift
//
//  ViewController.swift
//  CM1
//
//  Created by Camille Guinaudeau on 31/10/2017.
//  Copyright © 2017 Camille Guinaudeau. All rights
    reserved.
//

import UIKit

class ViewController: UIViewController {


    @IBAction func actionBouton(_ sender: Any) {
    }
    @IBOutlet weak var bouton: UIButton!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the
            view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be
            recreated.
    }
```

Main.Storyboard

Lien entre l'interface et le code - Fichier source

Ouvrir le fichier Main.storyboard et y créer les éléments d'interfaces graphique

Ouvrez le en mode source code



Main.Storyboard

Lien entre l'interface et le code - Fichier source

```
<viewController id="BYZ-38-t0r" customClass="ViewController" customModule="tedst"
customModuleProvider="target" sceneMemberID="viewController">
   <view key="view" contentMode="scaleToFill" id="8bC-Xf-vdC">
    <rect key="frame" x="0.0" y="0.0" width="375" height="667"/>
    <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
   <subviews>
    <button opaque="NO" contentMode="scaleToFill" fixedFrame="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center" buttonType="roundedRect"
lineBreakMode="middleTruncation" translatesAutoresizingMaskIntoConstraints="NO" id="3gj-6o-KzJ">
      <rect key="frame" x="164" y="297" width="46" height="30"/>
      <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
      <state key="normal" title="Button"/>
     </button>
     <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
textAlignment="natural" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines"
adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id="tVp-tN-6gc">
      <rect key="frame" x="166" y="225" width="42" height="21"/>
      <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
      <fontDescription key="fontDescription" type="system" pointSize="17"/>
      <nil key="textColor"/>
      <nil key="highlightedColor"/>
     </label>
    </subviews>
   <color key="backgroundColor" red="1" green="1" blue="1" alpha="1" colorSpace="custom"
customColorSpace="sRGB"/>
  <viewLayoutGuide key="safeArea" id="6Tk-OE-BBY"/>
 </view>
</viewController>
```


Main.Storyboard

Lien entre l'interface et le code - Fichier source

Ajoutez dans le fichier `ViewController.swift` les lignes de code pour déclarer les variables et les méthodes

```
@IBOutlet weak var label: UILabel!
@IBOutlet weak var bouton: UIButton!

@IBAction func actionBouton(_ sender: Any) {

}
```

ViewController.swift

Main.Storyboard

Lien entre l'interface et le code - Fichier source

Modifier le source code du fichier Main.Storyboard pour créer les liens (connections) entre les variables et les éléments d'interfaces graphiques

Le champ `destination` des connections doit être égal aux champs `id` des objets

```
<label opaque="NO" userInteractionEnabled="NO" contentMode="left"
horizontalHuggingPriority="251" verticalHuggingPriority="251" fixedFrame="YES" text="Label"
textAlignment="natural" lineBreakMode="tailTruncation" baselineAdjustment="alignBaselines"
adjustsFontSizeToFit="NO" translatesAutoresizingMaskIntoConstraints="NO" id=« tVp-tN-6gc »>

<button opaque="NO" contentMode="scaleToFill" fixedFrame="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center"
buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="3gj-6o-KzJ">

</view>
 <connections>
  <outlet property="bouton" destination="3gj-6o-KzJ" id="584-SC-Dge"/>
  <outlet property="label" destination="tVp-tN-6gc" id="u7C-hO-yUE"/>
 </connections>
 </viewController>
```

Main.storyboard

Main.Storyboard

Lien entre l'interface et le code - Fichier source

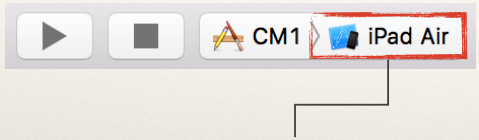
Modifier le source code du fichier Main.Storyboard pour créer les liens (connections) entre les variables et les éléments d'interfaces graphiques

```
<button opaque="NO" contentMode="scaleToFill" fixedFrame="YES"
contentHorizontalAlignment="center" contentVerticalAlignment="center"
buttonType="roundedRect" lineBreakMode="middleTruncation"
translatesAutoresizingMaskIntoConstraints="NO" id="3gj-6o-KzJ">
 <rect key="frame" x="164" y="297" width="46" height=« 30"/>
 <autoresizingMask key="autoresizingMask" flexibleMaxX="YES" flexibleMaxY="YES"/>
 <state key="normal" title="Button"/>
 <connections>
   <action selector="actionBouton:" destination="BYZ-38-t0r" eventType="touchUpInside"
id="Xm9-Rj-KMr"/>
 </connections>
</button>
```

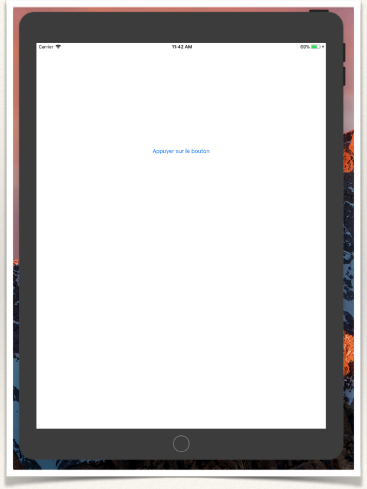
Main.storyboard

Xcode

iOS Simulator



Appareil sur lequel va être lancée la simulation



Plan du cours

1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

Plan du cours

1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

Variables

Les variables et opérateurs

Les variables

```
var entier = 2
var chaine = "Hello world !!"
var booleen = true
var float = 2.5
```

Les constantes

```
let constante = 3.14
```

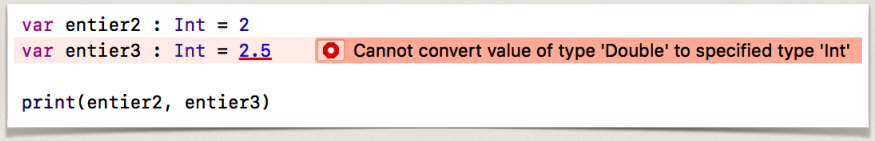
Les opérateurs

```
let addition = 1 + 2 // addition contiendra alors 3
let soustraction = 10 - 5 // soustraction contiendra alors 5
let multiplication = 6 * 7 // multiplication contiendra alors 42
let division = 100 / 11 // division contiendra alors 9
let modulo = 100 % 11 // division contiendra alors 1
```


Variables

Les variables et opérateurs

```
/* Association d'un type
précis à une variable */
var entier2 : Int = 2
var entier3 : Int = 2.5

print(entier2, entier3)
```




```
var entier: Int
var float: Float
var chaine: String
var booleen: Bool
```


Variables

Variables optionnelles

Nouvelle fonctionnalité de Swift.

Les variables optionnelles sont utilisées lorsqu'une variable est susceptible de n'avoir aucune valeur
—> c'est-à-dire pour une variable qui peut avoir soit une valeur ou pas de valeur du tout (nil)

Variables

Variables optionnelles

Une variable est dite optionnelle si lors de l'affectation on peut se retrouver avec deux cas :

- Une valeur peut être affectée.
- Une valeur ne peut pas être affectée.

```
let nombreString: String = "123"
let nombreInt = Int(nombreString)
```

`nombreInt` peut ne pas être affectée, elle est de type `Int?`

Variables

Variables optionnelles

Si vous êtes sûr que l'optionnelle a une valeur, alors vous pouvez utiliser le point d'exclamation (!) immédiatement après le nom de la variable pour forcer le changement de son type.
—> Conversion du type de l'optionnelle en type définitif

```
var newString : String?
newString = "XYZ" // newString de type String?
newString! // newString est maintenant de type String
```


Variables

Variables optionnelles

Utilisation du point d'exclamation sur une variable optionnelle qui n'a aucune valeur —> `« Fatal error: Unexpectedly found nil value while unwrapping an Optional value »`

Utiliser l'instruction if pour savoir si une optionnelle contient une valeur ou non

```
if newString != nil {
   print("La valeur de newString est : \(newString!).")
}
else {
   print("newString ne contient aucune valeur ")
}
```


Variables

Variables optionnelles

Mécanisme Optional Bindings
—> Vérification de la présence d'une valeur à l'intérieur de l'optionnelle et affectation de son contenu à une variable ou une constante temporaire en une seule instruction

```
if let actualNumber = possibleNumber.toInt() {
    print("\(possibleNumber) est un entier pour \
(actualNumber)")
}
else {
    print("La valeur \(possibleNumber) ne peut pas être
convertie")
}
```


Variables

Variables optionnelles implicites

Parfois, il est possible qu'une optionnelle ait toujours une valeur après sa déclaration
—> optionnelles implicites

```
var implicitInt : Int! // implicitInt = nil par défaut
```

La déclaration de type ! (optionnelle implicite) est similaire à ? (optionnelle classique)
Les deux sont des variables optionnelles, mais le ! est implicitement dévoilé, ce qui signifie que vous n'avez pas à le déballer pour accéder à sa valeur (mais il peut encore être nil).

Variables

Variables optionnelles implicites - The Nil-Coalescing Operator

L'opérateur nil-coalescing (`a ?? b`) « déballe » la variable optionnelle `a` si elle contient une valeur, ou retourne la valeur par défaut `b` si `a` est `nil`

La variable `a` est forcément de type optionnel

Le type de `b` doit correspondre au type stocké dans la variable `a`

Si la valeur de `a` est non nulle alors `b` n'est pas évaluée

Exemple

```
let defaultColorName = "red"
var userDefinedColorName: String?   // defaults to nil

var colorNameToUse = userDefinedColorName ?? defaultColorName
```


Les chaînes de caractères

Chaine de caractères

```
let nombre = 7
print("Le chiffre que vous avez choisi est \(nombre) !")
```

Opérateur de concaténation : +

```
let hello = "Hello,"
let world = " World !"
let helloWorld = hello + world
```

Conversion en String

```
let nombre1 = 1
let nombre2 = 2
let nombre = String(nombre1) + String(nombre2)
```


Les conditions et les boucles

if, else if, else

```
let n = 12
if (n >= 0 && n < 10) {
    // ...
}
else if (n >= 10 && n < 12) {
    // ...
}
else if (n == 12 || n == 13) {
    // ...
}
else if (n >= 14 && n <= 16) {
    // ...
}
else {
    // ...
}
```

condition ternaire

```
let note = 12
var moyenne: Bool

if note < 10 {
    moyenne = false
} else {
    moyenne = true
}

moyenne = note < 10 ? false : true
```


Les conditions et les boucles

switch

```
switch n {
case 0,1,2,3,4,5,6,7,8,9:
    // ...

case 10, 11:
    // ...

case 12, 13:
    // ...

case 14, 15, 16:
    // ...

default:
    // ...
}
```

switch

```
switch n {
case 0...9:
    // ...

case 10..<12:
    // ...

case 12, 13:
    // ...

case 14...16:
    // ...

default:
    // ...
}
```

Les conditions et les boucles

```swift
for i in 0...10 {
    print(i) // affichage de 0 à 10
}

for j in 0..<10 {
    print(j) // affichage de 0 à 9
}

for k in stride(from: 0, through: 10, by: 1) {
    print(k) // affichage de 0 à 10
}

for h in stride(from: 0, to: 10, by: 1) {
    print(h) // affichage de 0 à 9
}
```


Les conditions et les boucles

```swift
var nbDeLignes: Int = 1
while nbDeLignes <= 10 {
     print(nbDeLignes)
     nbDeLignes = nbDeLignes + 1
}


var nbDeLignes2: Int = 1
repeat {
    print(nbDeLignes2)
    nbDeLignes2 = nbDeLignes2 + 1
} while nbDeLignes2 <= 10
```


Structure de données

Les tuples

Les tuples

```swift
let tuple = (valeur1, valeur2, ...)
// Ou
var tuple = (valeur1, valeur2, ...)
```

—> Pas de limitation sur le nombre de valeurs ou le type

Récupération des valeurs

```swift
let tuple = (123, "cent-vingt-trois")
let (valNum, valString) = tuple
let (_, valString) = tuple
```

Possibilité de donner un nom aux valeurs

```swift
let tuple = (valNum:123, valString:"cent-vingt-trois")
let v = tuple.valNum
let s = tuple.valString
```


Structure de données

Les tableaux

Déclaration d'un tableau

```swift
let tableau = [valeur1, valeur2, ..., valeurN]
var tableauEntiers: [Int]
var prenoms = ["Marie", "Maxime", "Antoine", "Lucie",
"Mathilde"]
```

Récupération d'une valeur

```swift
let p1 = prenoms[0]
let p5 = prenoms[5]
    --> fatal error: Array index out of range
```

Ajout d'une valeur

```swift
prenoms = prenoms + ["Jean"]
prenoms += ["Jean"]

prenoms = ["Jean"] + prenoms
```

Structure de données

Les dictionnaires

Déclaration d'un dictionnaire

```
var personne = ["Nom": "Durand", "Prénom": "Maxime", "Adresse":
"94 rue machin", "Ville": "Lille"]

var dictionnaire: [String: Int]
```

Récupération d'une valeur à partir d'une clé

```
let nom = personne["Nom"]
```

Ajout d'un nouveau couple clé/valeur

```
personne["Commentaire"] = "Personne super cool !"
```

Modification d'une valeur existante

```
personne["Adresse"] = "12 rue bidule"
```


Structure de données

Parcours

Tableau

```
let prenoms = ["Marie", "Maxime", "Antoine", "Lucie",
                « Mathilde"]


for i in stride(from: 0, to: prenoms.count, by: 1) {
    print("- " + prenoms[i])
}
```

```
- Marie
- Maxime
- Antoine
- Lucie
- Mathilde
```

```
let personne = ["Nom": "Durand", "Prénom": "Maxime", "Adresse":
"94 rue machin", "Ville": "Lille"]

for (cle, valeur) in personne {
    print(cle + " - " + valeur)
}
```

```
Ville - Lille
Nom - Durand
Prénom - Maxime
Adresse - 94 rue machin
```

Dictionnaire

Structure de données

tableaux et dictionnaires

count —> nombre d'éléments du tableau ou dictionnaire.

append() —> ajout d'un nouvel élément à la fin du tableau à la condition que cet élément respecte le type de ce dernier

attribut first ou attribut last —> valeur du premier ou du dernier élément du tableau

isEmpty() —> vrai si le tableau ou le dictionnaire est vide, faux sinon

insert() —> insert un nouvel élément dans le tableau ou le dictionnaire. Prend en paramètre le nouvel élément et l'index où placer ce nouvel élément

removeAll() —> supprime tous les éléments du tableau ou du dictionnaire

removeAtIndex() —> supprime l'élément du tableau à l'index donné en paramètre

removeLast() —> supprime le dernier élément du tableau

reverse() —> inverse l'ordre des éléments du tableau

Les fonctions

Déclaration et utilisation d'une fonction

```
// On déclare les fonctions
func disBonjour() {
    print("Bonjour !")
}

func disBonjourCommentCaVa() {
    disBonjour()
    print("Comment ça va ?")
}

// Enfin, on peut s'en servir !
disBonjourCommentCaVa()
```

Les fonctions

Fonctions à plusieurs paramètres

```
func nomDeLaFonction(parametre1: Type, parametre2: Type) {
   // Instructions
}
```

Le nom des paramètres doivent apparaître lors de l'appel

```
nomDeLaFonction(parametre1:p1, parametre2:p2)
```

```
func nomDeLaFonction(parametre1: Type, parametre2: Type) ->
TypeRetour {
     // Instructions
     return laValeurARetourner
}
```


Plan du cours

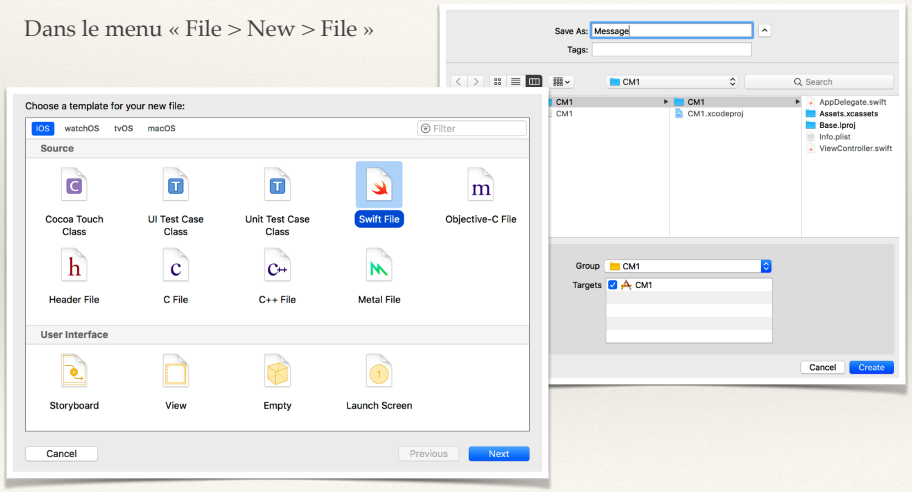
1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

Plan du cours

1. Swift et Cocoa Touch
2. IDE Xcode
3. Langage Swift
4. Création de classe

Création d'une nouvelle classe

Dans le menu « File > New > File »



Programmation Orientée-Objet

Rappels

Les classes et instances d'objet

Chaque objet est d'un certain type qui correspond à sa classe
—> Plusieurs objets font partie de la même classe et ont le même comportement défini par des méthodes

Ces méthodes sont envoyées à des objets individuels qui sont les instanciations d'une classe, ce sont des instances d'objet

La classe `Message` permet d'instancier un message en particulier qui va avoir un contenu, un destinataire et un expéditeur particulier

Programmation Orientée-Objet

Rappels

Les variables d'instances

Les instances possèdent des variables dont les valeurs sont initialisées et/ou modifiées par des méthodes

Ces valeurs sont propres à chaque instance

Par exemple la classe `Message` possède des variables d'instance `contenu`, `destinataire` et `expediteur` dont les valeurs seront spécifiques à chaque instance

Variables d'instance

```swift
//  Message.swift

import Foundation

class Message {
    public var contenu : String
    internal var destinataire : String
    private var expediteur : String
}
```


Méthodes

```swift
//  Message.swift

import Foundation

class Message {
    public var contenu : String
    internal var destinataire : String
    private var expediteur : String


    func rediger(contenu : String) {
        self.contenu = contenu
    }

    func ecrire() {
        print("Bonjour"+self.destinataire)
        print("Voici votre message : "+self.contenu)
        print("Signé : "+self.expediteur)
    }
}
```

Création d'une instance de classe

Constructeurs

Chaque classe défini ou hérite d'**au moins un** constructeur

Les constructeurs s'appellent `init` et possèdent plus ou moins de paramètres

Chaque classe possède au moins un « Designated initializer »

- Constructeur principal qui doit être appelé lors de l'initialisation d'une instance de la classe
- Il doit être utilisé dans tous les autres constructeurs
- Un constructeur désigné doit appeler le constructeur désigné de sa super-classe

Création d'une instance de classe

Constructeurs

Chaque classe peut posséder un « Convenience initializer »

- Permet d'initialiser les variables d'instances avec des valeurs par défaut

- Un « convenience initializer » doit appeler un constructeur de la même classe
- Un « convenience initializer » doit **dans l'idéal** appeler un « designated initializer »

Création d'une instance de classe

Constructeurs

```
//  Message.swift

import Foundation

class Message {
    public var contenu : String
    internal var destinataire : String
    private var expediteur : String

    init(contenu : String, destinataire : String, expediteur: String) {
        self.contenu = contenu
        self.destinataire = destinataire
        self.expediteur = expediteur
    }

    convenience init() {
        self.init(contenu : "", destinataire : "inconnu",
                  expediteur: "inconnu")
    }
}
```


Création d'une instance de classe

Affichage d'un objet

```
//  Message.swift

import Foundation

class Message {
    var contenu : String
    var destinataire : String
    var expediteur : String
    var description: String {
        return "Message : \(contenu)\nà : \(destinataire)\n
          de : \(expediteur)"
    }

    init(contenu : String, destinataire : String, expediteur: String) {
        self.contenu = contenu
        self.destinataire = destinataire
        self.expediteur = expediteur
    }

    //  ...

}
```

Utilisation d'une classe

```swift
//  ViewController.swift

import UIKit

class ViewController: UIViewController {

    var tabMessage : [Message] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
                        typically from a nib.
        let m = Message()
        tabMessage.append(m)
        for message in tabMessage {
            print(message.description)
        }
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()

    }
}
```

```
Message :
à : inconnu
de : inconnu
```


Variables

Variables « lazy »

Mécanisme intégré dans le langage permettant de calculer des variables uniquement si cette variable est appelée

```swift
class Message {
    public var contenu : String
    internal var destinataire : String
    private var expediteur : String
    var description: String {
        return "Message : \(contenu)\nà : \(destinataire)\nde : \(expediteur)"
    }

    lazy var contenuCrypte : Int = {
        return cryptage(of:self.contenu)
    }()

    func cryptage(of : String) -> Int {
        // transformation du contenu
        return 0
    }
}
```

Si la variable n'est jamais appelée, la fonction n'est jamais exécutée, ce qui permet d'économiser du temps de traitement

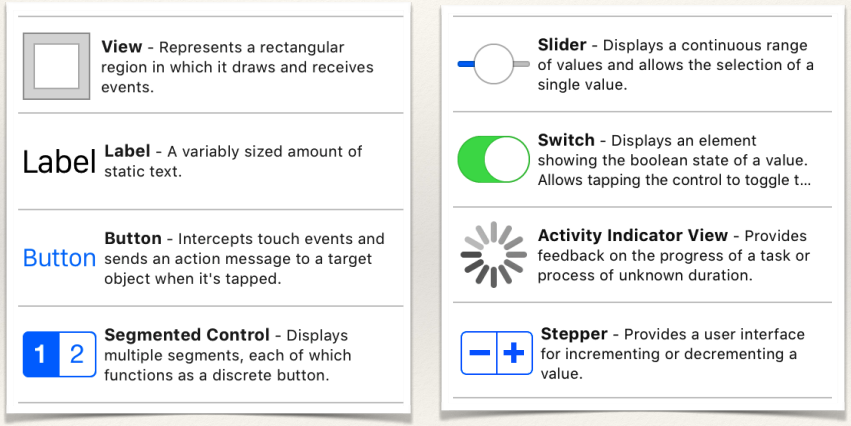
Inscription sur Moodle

Développement multi-support —>
Développement d'applications mobiles iOS

Clé d'inscriptions :

- iOS-LPPRISM-1
- iOS-LPPRISM-2

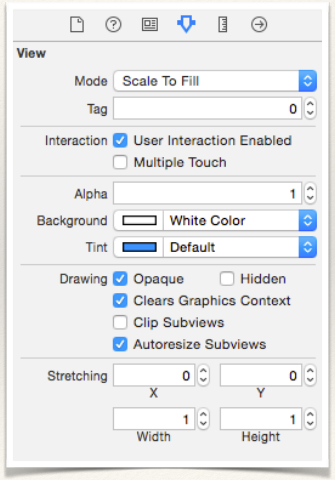
Éléments d'interface graphique



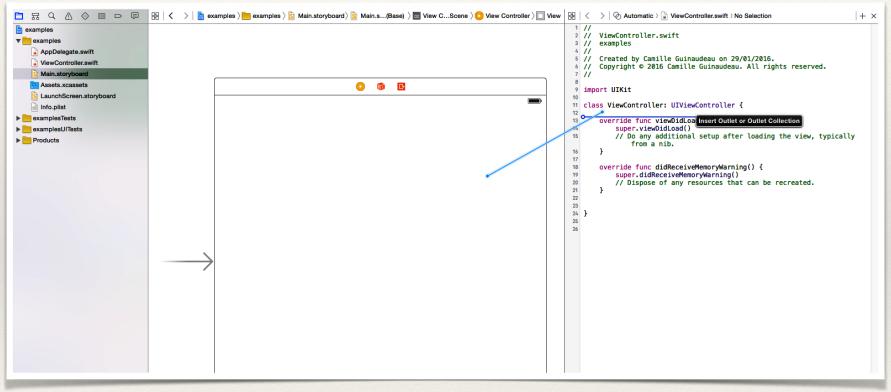
UIView

Les propriétés d'un objet de type `UIView` sont :

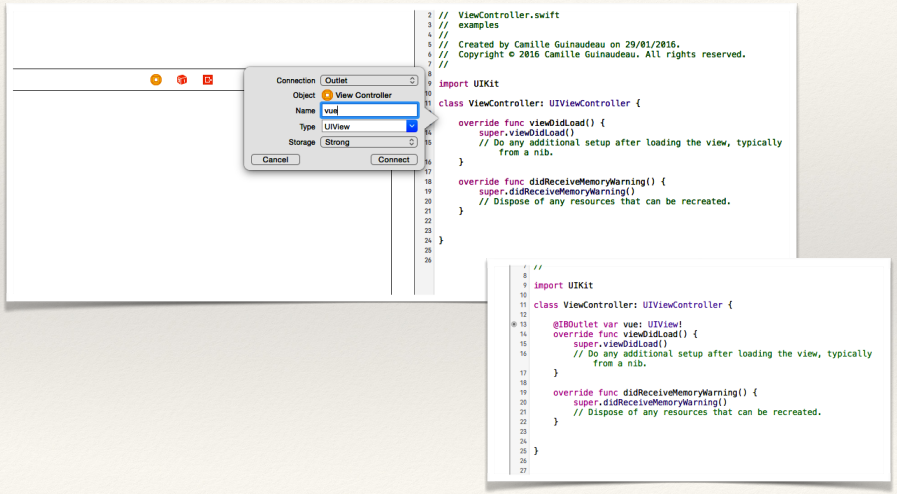
- mode
- tag
- interaction
- alpha / background / tint
- drawing
- stretching



UIView



UIView




```swift
//  ViewController.swift
//  examples
//
//  Created by Camille Guinaudeau on 29/01/2016.
//  Copyright © 2016 Camille Guinaudeau. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically
            from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```


UIView

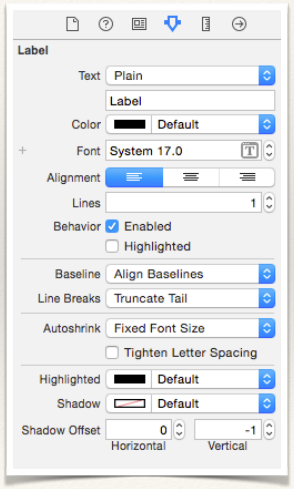
ViewController.swift

```swift
import UIKit

class ViewController: UIViewController {

    @IBOutlet var vue: UIView!

override func viewDidLoad() {
        super.viewDidLoad()

        self.vue.backgroundColor = UIColor.red;
        self.vue.tintColor = UIColor.orange;
        self.vue.alpha = 0.5;
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

UILabel

Les propriétés d'un objet de type `UILabel` sont :

- text (Plain / Attributed)
  color / font / alignment / lines
- enabled
- highlighted
- baseline / line break / autoshrink
- shadow (couleur et position)

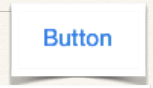


UILabel

ViewController.swift

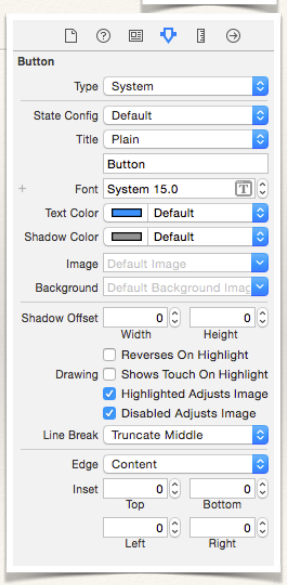
```swift
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var label: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()

        self.label.text = "Texte à afficher";
        self.label.isHighlighted = true;
        self.label.font = UIFont(name:"AmericanTypewriter", size: 12.0);

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```


UIButton

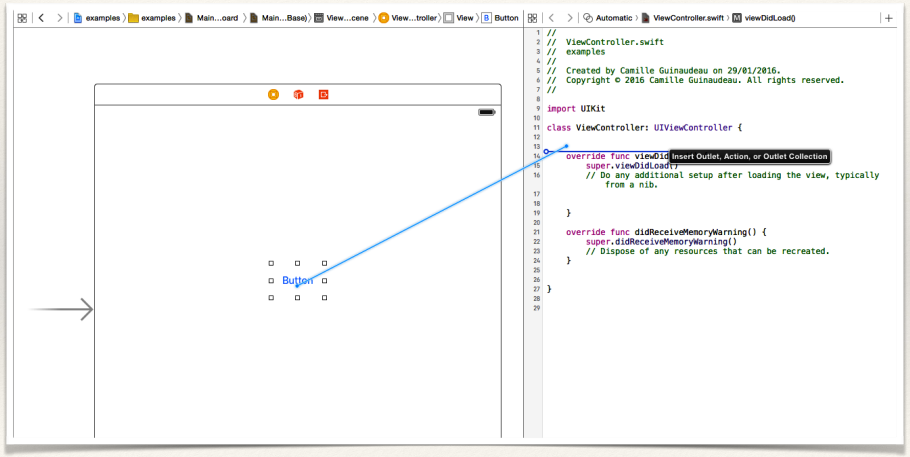


Les propriétés d'un objet de type `UIButton` sont :

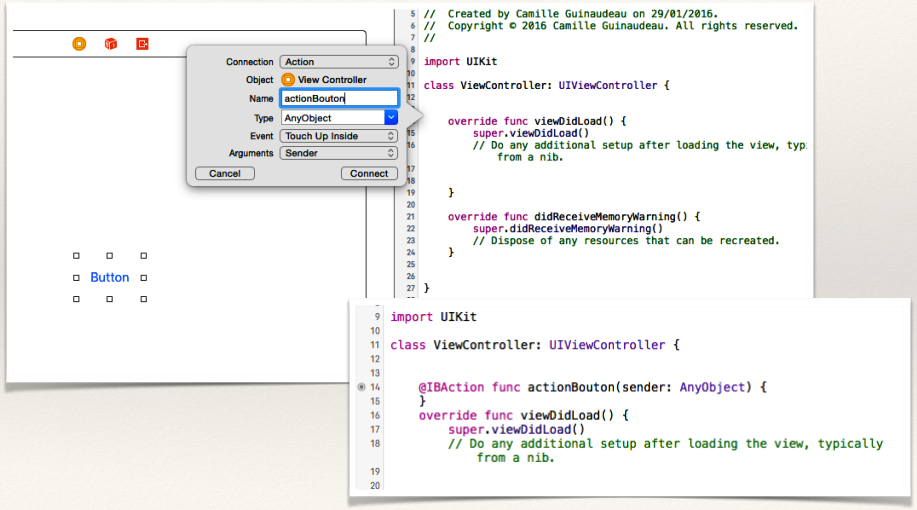
- type (system / custom / infos / add contact)
- state config
  (default / highlighted / selected / deselected)
- title (plain / attributed)
- font / text color / shadow color / image / background
- drawing
- line break
- edge / inset



UIButton



UIButton



UIButton

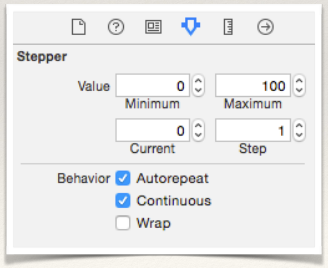
ViewController.swift

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var bouton: UIButton!
    @IBAction func actionBouton(sender: AnyObject) {
        print("L'utilisateur a appuyé sur le bouton :
                    \(self.bouton.currentTitle!)");
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        self.bouton.backgroundColor = UIColor.red;
        self.bouton.setTitle("OK", forState: UIControlState.normal);

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```


UIStepper

Les propriétés d'un objet de type `UIStepper` sont :

- value
- minimumValue
- maximumValue
- stepValue
- autorepeat
- continuous
- wraps



UIStepper

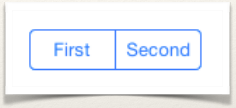
ViewController.swift

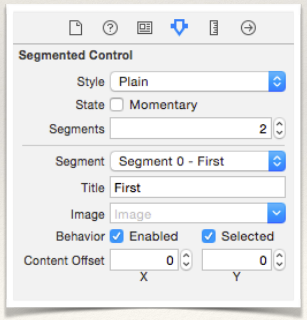
```
import UIKit

class ViewController: UIViewController {


    @IBOutlet weak var stepper: UIStepper!
    override func viewDidLoad() {
        super.viewDidLoad()

        self.stepper.stepValue = 5.0;
        self.stepper.isContinuous = true;
        self.stepper.autorepeat = true;

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

UISegmentedControl

Les propriétés d'un objet de type `UISegmentedControl` sont :

- style (Plain / Bordered / Bar) - obsolète depuis iOS7.0
- state (Momentary)
- segments
- behavior (enabled / selected)
- content offset





UISegmentedControl

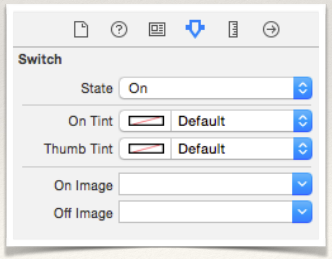
ViewController.swift

```swift
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var segmentedControl: UISegmentedControl!
    @IBAction func actionSegmentedControl(sender: AnyObject) {
        if (self.segmentedControl.selectedSegmentIndex == 0) {
            print("L'utilisateur a sélectionné la première option");
        }
        else {
            print("L'utilisateur a sélectionné la seconde option");
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```


UISwitch

Les propriétés d'un objet de type `UISwitch` sont :

- state (On / Off)
- on tint
- thumb tint
- on image
- off image



UISwitch

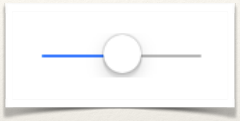
ViewController.swift

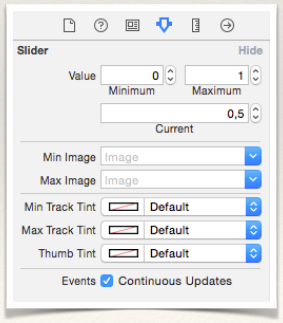
```swift
import UIKit

class ViewController: UIViewController {


    @IBOutlet weak var switche: UISwitch!
    @IBAction func actionSwitch(sender: AnyObject) {
        if (self.switche.isOn) {
            print("Le bouton switch est positionné sur ON");
        }
        else {
            print("Le bouton switch est positionné sur OFF");
        }
    }
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

UISlider

Les propriétés d'un objet de type `UISlider` sont :

- minimumValue
- maximumValue
- value
- minimum/maximum image
- minimum/maximum track tint
- thumb tint
- continuous update





UISlider

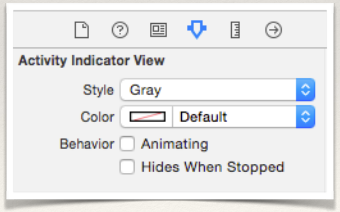
ViewController.swift

```
import UIKit

class ViewController: UIViewController {


    @IBOutlet weak var slider: UISlider!
    @IBAction func actionSlider(sender: AnyObject) {
        print("%f", self.slider.value);
    }
    override func viewDidLoad() {
        super.viewDidLoad()

        self.slider.isContinuous = true;
        self.slider.minimumValue = 10.0;
        self.slider.maximumValue = 20.0;
        self.slider.value = 15.0;
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```


UIActivityIndicatorView

Les propriétés d'un objet de type `UIActivityIndicatorView` sont :

- style (white/large white/gray)
- color
- animating
- hides



UIActivityIndicatorView

ViewController.swift

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var activityIndicator: UIActivityIndicatorView!
    override func viewDidLoad() {
        super.viewDidLoad()

        self.activityIndicator.hidesWhenStopped = true;
        self.activityIndicator.startAnimating();
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

La méthode `stopAnimating()` et l'attribut `isAnimating` permettent d'arrêter l'animation et de tester si l'objet `UIActivityIndicatorView` est animé ou non