

# Cours IOS

Octobre 2019



2

## Introduction

Introduction – Programme – Planning - Liste des TP- Evaluation

# Introduction

3

▣ Guillaume Gonzales

▣ Tokidev S.A.S.

■ Consulting, Bureau d'étude,  
développement  
informatique.

■ [www.tokidev.fr](http://www.tokidev.fr)



# Avant de commencer

- ▣ Posez des questions !

- Si un point ne vous semble pas clair, n'hésitez pas à me poser des questions.

- ▣ Me contacter en dehors du cours

- Si vous avez besoin d'aide en dehors des horaires de cours, contactez moi par e-mail : [gonzales.guillaume@tokidev.fr](mailto:gonzales.guillaume@tokidev.fr)
- Pour le cours, l'examen ou le projet.

# Avant de commencer

5

- ▣ Google est votre ami !
  - Si vous rencontrez un problème, ayez le réflexe de chercher sur internet... la réponse y sera la plupart du temps !
- ▣ Le debugger est votre ami !
  - Nous verrons dans ce cours comment utiliser le debugger. Couplez aux recherches internet, vous pourrez résoudre ainsi tous vos problèmes.

# Programme

6

- ▣ Présentation générale
- ▣ Récapitulatif : Langage C
  - Opérateurs
  - Format des Strings
  - Variables
  - Conditions
  - Boucles
  - Pointeurs

- ▣ Objective-C
  - POO
  - Syntaxe
  - Appel de fonction
  - Self
  - Interface & Implémentation
  - Accesseurs
  - ARC

# Programme

7

## ▣ Swift

- Présentation générale
- Commentaires
  - Variables - Constantes
  - Print
  - Arrays – Dictionaries
  - Conditions – Boucles
  - Fonctions
  - Classes
  - Constructeurs

# Programme

8

## ■ iOS

- Présentation générale
- Xcode - Structure d'un projet
- Debugger
- Interface Utilisateur
  - MVC
  - Storyboard
  - Vues & Conteneur
  - Delegate
  - Notification Center
  - Composants & Événements
  - Auto Layout

## ■ Web Services

## ■ Données

- Librairie externe
- Parsing XML / JSON
- Core Data

## ■ Media

- Caméra
- Audio
- Vidéo
- Géolocalisation

## ■ Et maintenant ...



# Évaluation

- ▣ Projet en TP

- ▣ Application iPhone

- ▣ Partiel

- ▣ Questions générales (QCM)
  - ▣ Questions ouvertes
  - ▣ Code à compléter
  - ▣ Code à produire



10

## Présentation Générale

Objective C / iOS

# Objective C

11

- ▣ Apparue en 1983
- ▣ Conçu par Brad Cox et Tom Love
- ▣ Utilisation principale :
  - Apple OS X
  - Apple iOS
- ▣ Langage orienté objet
- ▣ Influencé par le :
  - C
  - SmallTalk

# Apple iOS

12

- Système d'exploitation de l'iPhone / iPad
- Mars 2008 pour iPhone OS
- Renommé iOS en Juin 2010



# Apple iOS 10

13

- Septembre/Octobre 2016
- Nouveau langage : Swift





14

## Récapitulatif Langage C

Opérateurs - Format des Strings - Variables - Conditions – Boucles – Pointeurs

# Opérateurs

Operator	Meaning	Example
<code>==</code>	Equal to	<code>count == 10</code>
<code>!=</code>	Not equal to	<code>flag != DONE</code>
<code>&lt;</code>	Less than	<code>a &lt; b</code>
<code>&lt;=</code>	Less than or equal to	<code>low &lt;= high</code>
<code>&gt;</code>	Greater than	<code>points &gt; POINT_MAX</code>
<code>&gt;=</code>	Greater than or equal to	<code>j &gt;= 0</code>

# Incrémenter & Décrémenter

Operator	Meaning	Example
++	Add one to variable	<code>++count;</code> <code>i++;</code>
--	Subtract one from variable	<code>--index;</code> <code>timer--;</code>



# Format des Strings

Exemple :

```
printf("int  
value =  
%i",someInt);
```

```
NSLog("A  
string =  
%@",aString);
```

Type	Constants	NSLog
char	'a', '\n'	%c
short int	-	%hi, %hx, %ho
unsigned short int	-	%hu, %hx, %ho
int	12, -97, 0xFFE0, 0177	%i, %x, %o
unsigned int	12u, 100U, 0XPFu	%u, %x, %o
long int	12L, -2001, 0xffffL	%li, %lx, %lo
long long int	0xe5e5e5e5LL, 5001l	%lli, %llx, %llo
unsigned long long int	12ULL, 0xffeeULL	%llu, %llx, %llo
float	12.34f, 3.1e-5f	%f, %e, %g
double	12.34, 3.1e-5	%f, %e, %g
long double	12.34l, 3.1e-5l	%Lf, %Le, %Lg
id	-	%p

# Variables

```
int a = 5;  
int b = 4;  
int c = 12;  
int x;
```

```
x = a + b + c;
```

# Conditions – If/Else

```
int isThere;  
isThere = 1;  
  
if(isThere == 1)  
{  
    printf("Something is there\n");  
}  
else  
{  
    printf("Nothing here\n");  
}
```

# Tableaux

```
int numbers[5];
```

```
numbers[0] = 1;  
numbers[1] = 2;  
numbers[2] = 3;  
numbers[3] = 4;  
numbers[4] = 5;
```

```
printf("numbers[0] = %i\n", numbers[0]);  
printf("numbers[1] = %i\n", numbers[1]);  
printf("numbers[2] = %i\n", numbers[2]);  
printf("numbers[3] = %i\n", numbers[3]);  
printf("numbers[4] = %i\n", numbers[4]);
```

# Boucles - For

```
int numbers[5];
```

```
numbers[0] = 1;  
numbers[1] = 2;  
numbers[2] = 3;  
numbers[3] = 4;  
numbers[4] = 5;
```

```
for(int x=0;x<=5;++x)  
{  
    printf("x=%i\n",numbers[x]);  
}
```

```
int i = 0;  
while (i <= 5)  
{  
    printf("i=%i\n",i);  
    i++;  
}  
i = 0;  
do  
{  
    printf("i=%i\n",i);  
    i++;  
} while (i <= 5);
```

# Switch

```
int    sdkVersion    =    1;

switch    (sdkVersion)
{
    case    2:
        printf("SDK Version 2.23");
        break;
    case    1:
        printf("SDK Version 1.332");
        break;
    default:
        printf("Unknown SDK Version!");
        break;
}
```

# Fonctions

```
void    displayVersion(int ver)
{
    printf("Version %i", ver);
}
```

# Pointeurs

24

▣ Un **pointeur** est une adresse mémoire

■ *type \*variableName;*



# Pointeurs

25

- ▣ Déclare le pointeur d'un type Entier

▣ `int *foo_ptr;`

- ▣ Déclare un entier

▣ `int foo1 = 1;`

- ▣ Attribut l'adresse de foo1 dans foo\_ptr

▣ `foo_ptr = &foo1;`

# Pointeurs

26

```
int *foo_ptr;
```

```
int foo1;
```

```
foo1 = 1;
```

```
foo_ptr = &foo1;
```

```
printf("foo_ptr = %p\n", foo_ptr);
```

```
printf("foo1 = %p\n", &foo1);
```

```
printf("*foo_ptr = %d\n", *foo_ptr);
```

```
*foo_ptr = (*foo_ptr)+1;
```

```
printf("foo1 = %d", foo1);
```

```
foo_ptr = 0x7fff6eeb6b8c
foo1 = 0x7fff6eeb6b8c
*foo_ptr = 1
foo1 = 2
```



27

## Objective-C

POO - Syntaxe - Appel de fonction - Self - Interface & Implémentation – Accesseurs

# POO

28

- ▣ Classe

- Définition des données, types et comportement

- ▣ Instance

- Une occurrence unique d'une classe

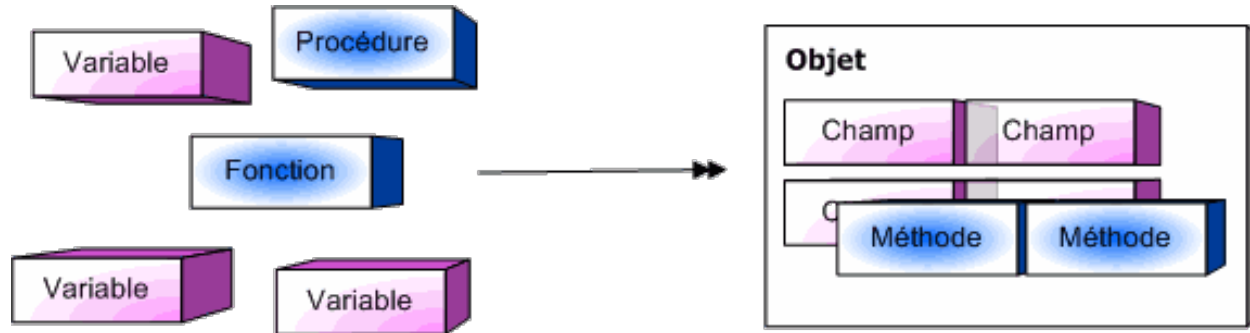
- ▣ Objet

- Une instance d'une classe donnée

# POO

## ■ Encapsulation

- Concept même de l'objet. Permet de réunir sous la même entité les données et les moyens de les gérer.

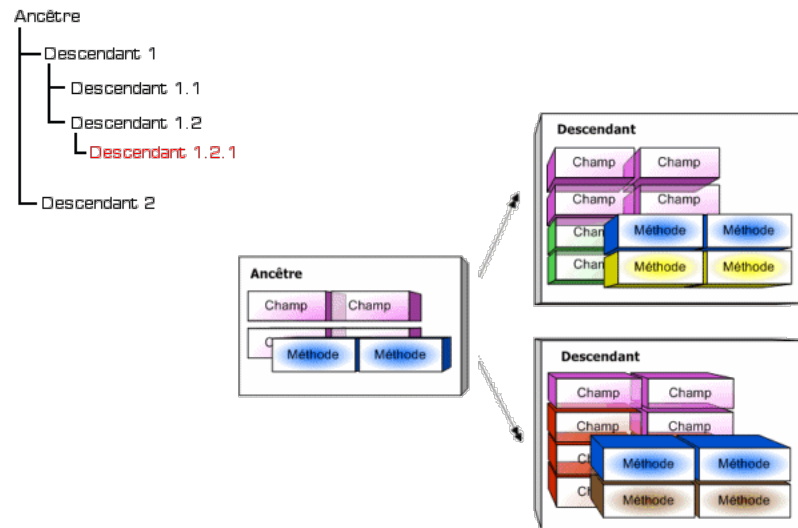


# POO

30

## ■ Héritage

- Permet à une classe de posséder les attributs et méthodes de sa classe mère. Elle peut également en posséder d'autres, propre à elle.



# POO

31

## ▣ Polymorphisme

- Permet à un objet de posséder plusieurs formes.
- Permet, en d'autres termes, au système de choisir dynamiquement la méthode qui correspond au type réel de l'objet en cours.

# Syntaxe

22

- ▣ Caractère [ - Touche *Alt* + *Shift* + (
- ▣ Caractère ] - Touche *Alt* + *Shift* + )
- ▣ Caractère { - Touche *Alt* + (
- ▣ Caractère } - Touche *Alt* + )
- ▣ Caractère \ - Touche *Alt* + *Shift* + /
- ▣ Caractère | - Touche *Alt* + *Shift* + L
- ▣ Caractère ~ - Touche *Alt* + N



# Appel de fonction

33

- ▣ Appeler une fonction d'un objet

```
[object function]  
[alert show];
```

# Appel de fonction

34

- Appeler une fonction d'un objet avec un argument

```
[alert show:@"now"];
```

```
[object function:@"argument"];
```

# Appel de fonction

35

- ▣ Appeler une fonction d'un objet avec plusieurs arguments

```
[alert show:@"now"
```

```
times:2
```

```
turnOff:YES];
```

# Appel de fonction

## ▣ Exemple complet :

```
UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"My Alert"  
message:@"Danger! Danger!"  
delegate:self  
cancelButtonTitle:@"Cancel?"  
otherButtonTitles:nil];  
  
[alert show];
```

# Self

- Utiliser pour appeler une méthode depuis l'objet courant.
- Equivalent de « **This** »

```
[self function];
```

# Interface & Implementation

38

## ▣ Classe

- Une classe est composée de deux fichiers :

- ▣ Un fichier .h (*Définition*)
- ▣ Un fichier .m (*Implémentation*)

## ▣ Directives de compilation

- Chaque fichier inclus une directive de compilation :

- ▣ @interface ... @end (.h)
- ▣ @implementation ... @end (.m)

# Interface & Implementation

## ▣ Interface (.h)

- Correspond à la définition de la classe

## ▣ Implémentation (.m)

- Correspond à l'implémentation de l'interface

# Interface & Implementation

## ▣ Exemple d'interface (.h)

```
@interface NewClassName : ClassWeInheritFrom
{
    DataType *privateVariable;
}
@property (nonatomic, retain) DataType *protectedVariable;

- (returnDataType) methodName:(dataType *) arg1;

@end
```



# Interface & Implementation

41

## ■ Exemple d'implémentation (.m)

```
@implementation NewClassName
@synthesize protectedVariable;

- (returnType)methodWeInherited:(DataType *)arg
{

}

- (returnDataType) methodName:(dataType *) arg1
{

}

@end
```

# Instance & Méthodes de classe

42

## ■ Exemple d'implémentation (.m)

@implementation

NewClassName

```
+ (id)getClass
{
    return self;
}
```

@end

# Instance & Méthodes de classe

43

```
+ (id) getClass { ... }
```

```
- (id) getString{ ... }
```

```
// Méthode de classe
```

```
someVar = [SomeClass getClass];
```

```
// Méthode d'instance
```

```
someClassInstance = [[SomeClass alloc] init];
```

```
someVar = [someClassInstance getString];
```

# Accesseurs

44

- ▣ Getter & Setter d'une variable de classe
- ▣ Peut-être créer à la main, en respectant la convention de nommage : **setMaVariable**
- ▣ Peut-être générer automatiquement, en utilisant **@synthesize**

# Accesseurs

45

▣ Exemple d'interface (.h) définissant une variable de classe.

```
@interface NewClassName : ClassWeInheritFrom
{

}

@property (nonatomic, retain) DataType *instanceVariable;

@end
```

# Accesseurs

46

- ▣ Exemple de début d'implémentation (.m)

```
@implementation NewClassName  
@synthesize instanceVariable;
```

- ▣ Le @synthesize permet donc d'utiliser :
  - [object setInstanceVariable:val]
  - [object instanceVariable]

# ARC

## ▣ Automatic Reference Counting

- ▣ Permet de s'affranchir de la gestion de mémoire, maintenant gérée par le compilateur.



48

## Language Swift

Variables – Arrays – Dictionaries – Conditions – Boucles – Functions - Classes



# Langage Swift

- ▣ Apparu en 2014 (Septembre)
- ▣ Inspiré par : Objective-C / Ruby / Python...
- ▣ Extension : .swift

# Commentaires

50

## ▣ Commentaire sur une ligne

```
// this is a comment
```

## ▣ Commentaire sur plusieurs lignes

```
/* this is also a comment,  
but written over multiple lines */
```

# Variables - Constantes

51

- ▣ Le type des variables/constantes n'est pas obligatoire.

```
let maximumNumberOfLoginAttempts = 10
```

- ▣ Pour définir une variable privée

```
var currentLoginAttempt = 0
```

```
private let maximumNumberOfLoginAttempts = 10
```

```
private var currentLoginAttempt = 0
```

# Types des variables

52

- ▣ Le type des variables/constantes n'est pas obligatoire.

```
let implicitInteger = 70
```

```
let implicitDouble = 70.0
```

```
let explicitDouble: Double = 70
```

# Conversion des variables

53

- Pas de conversion implicite.

```
let label = "The width is "
```

```
let width = 94
```

```
let widthLabel = label + String(width)
```

```
let apples = 3
```

```
let appleSummary = "I have \(apples) apples."
```

# Print

54

- ▣ Afficher le contenu d'une variable ou constante

```
println(friendlyWelcome)
```

```
println("This is a string")
```

```
println("The current value of friendlyWelcome is \"(friendlyWelcome)\"")
```

# Arrays - Dictionaries

55

```
var shoppingList = ["catfish", "water", "tulips", "blue paint"]  
shoppingList[1] = "bottle of water"
```

```
var occupations = [  
  "Malcolm": "Captain",  
  "Kaylee": "Mechanic",  
]
```

```
occupations["Jayne"] = "Public Relations"
```

```
let emptyArray = String[]()  
let emptyDictionary = Dictionary<String, Float>()
```

# Conditions - Boucles

## ▣ Flux de contrôles standards

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
```



# Conditions - Boucles

## ▣ Switch Case

```
let vegetable = "red pepper"

switch vegetable {
  case "celery":
    let vegetableComment = "Add some raisins and make ants
on a log."

    case "cucumber", "watercress":
      let vegetableComment = "That would make a good tea
sandwich."

    case let x where x.hasSuffix("pepper"):
      let vegetableComment = "Is it a spicy \(x)?"
  default:
    vegetableComment = "Everything tastes good in soup."
```

# Conditions - Boucles

58

## □ Do...while

```
var n = 2
```

```
while n < 100 {  
    n=n*2  
}
```

```
var m = 2
```

```
do {  
    m=m*2  
} while m < 100
```

# Conditions - Boucles

## ▣ For (itérations)

```
var firstForLoop = 0
```

```
for i in 0..3 {  
    firstForLoop += i  
}
```

```
var secondForLoop = 0
```

```
for var i = 0; i < 3; ++i {  
    secondForLoop += 1  
}
```

# Fonctions

60

## ▣ Sans paramètres

```
func greet(){  
    ...  
}
```

# Fonctions

61

## ▣ Avec paramètres

```
func greet(name: String, day: String) -> String {  
    return "Hello \ \(name), today is \ \(day)."  
}  
  
greet("Bob", "Tuesday")
```

# Fonctions

62

## ▣ Plusieurs valeurs de retour

```
func getGasPrices() -> (Double, Double, Double) {  
    return (3.59, 3.69, 3.79)  
}
```

```
getGasPrices()
```

# Classes

63

## □ Déclaration

```
class Shape {  
    var numberOfSides = 0  
  
    func simpleDescription() -> String {  
        return "A shape with \(numberOfSides) sides."  
    }  
}
```

# Classes

04

## ▣ Instantiation

```
var shape = Shape()
```

```
shape.numberoSides = 7
```

```
var shapeDescription = shape.simpleDescription()
```



# Classes

65

## ▣ Héritage

```
class NamedShape {  
    var numberOfSides: Int = 0  
    var name: String  
  
    init(name: String) {  
        self.name = name  
    }  
  
    func simpleDescription() -> String {  
        return "A shape with \$(numberOfSides) sides."  
    }  
}
```

# Classes

## □ Héritage

```
class Square: NamedShape {  
    var sideLength: Double  
  
    init(sideLength: Double, name: String) {  
        self.sideLength = sideLength  
        super.init(name: name)  
        numberOfSides = 4  
    }  
  
    func area() -> Double {  
        return sideLength * sideLength  
    }  
  
    override func simpleDescription() -> String {  
        return "A square with sides of length \"$(sideLength).\""  
    }  
}
```

# Classes

## ▣ Héritage

```
var test = Square(sideLength: 5.2, name: "my test square")  
test.area()  
test.simpleDescription()
```

# Constructeur

- ▣ Pour surcharger le constructeur (init) dans un ViewController
- ▣ Toujours initialiser vos variables de classe dans Swift !

```
required init(coder aDecoder: NSCoder) {  
    // Variables à initialiser  
    super.init(coder: aDecoder)  
}
```

# Protocol

## ▣ Déclaration d'un protocole

```
protocol ExampleProtocol {  
    var simpleDescription : String { get }  
    mutating func adjust()  
}
```

# Protocol

## ▣ Implémentation d'un protocole

```
class SimpleClass: ExampleProtocol {  
    var simpleDescription: String = "A very simple class."  
    var anotherProperty: Int = 69105  
  
    func adjust() {  
        simpleDescription += "  Now 100% adjusted."  
    }  
}
```

# Force-unwrap

71

## □ Force-unwrap

```
class Person {  
    var name: String ?  
}  
  
let person = Person()  
person.name = "John Doe"  
  
print(person.name)  
print(person.name!)
```