

Formation du langage Swift



Les bases du langage Swift

M^{lle} Kathia CHENANE

Plan

- Présentation de Swift
 - Swift
 - Historique
 - Comparaison avec l'Objective C
- Installation des outils de développement
 - Installation Xcode
 - Utiliser Xcode
 - Programmer en Swift avec Xcode
- Les variables et opérations
 - Exercice Pratique 1

Plan

- Les conditions
 - Exercice Pratique 2
- Les boucles
 - Exercice Pratique 3
- Les tableaux et les dictionnaires
 - Exercice Pratique 4
- Les fonctions
 - Exercice Pratique 5
- TP

Présentation du langage Swift ^{1/3}

Swift

- Langage de programmation créé par Apple en 2014
- Créé pour élaborer des applications sous le système d'application iOS
- Mise à jour de Swift en passant de la version de 2.0 à 3.1
- Langage plus rapide comme son nom l'indique

Présentation du langage Swift ^{2/3}

Historique

- Développement de Swift a commencé avec Chris Lattner un superviseur du département de « developer Tools » chez Apple
- Lancement le 2 juin 2014

Présentation du langage Swift ^{3/3}

Comparaison avec l'Objective C

- Inspiré de l'Objective C, C#, Python
- Plus intuitif et rapide
- Parmi les changements de syntaxe :
 - Condition de if sans parenthèses avec une accolade à droite.
 - Disparition du break pour Switch
 - Le mot func pour définir une fonction.
 - Point-virgule optionnel.

Outils de développement

Xcode 8.2

Télécharger Xcode sous Apple store

Xcode

Par Apple

Les Indispensables

Ouvrez le Mac App Store pour acheter et télécharger des apps.



Afficher dans le Mac App Store

Description

Xcode includes everything developers need to create great applications for Mac, iPhone, iPad, Apple TV, and Apple Watch. Xcode provides developers a unified workflow for user interface design, coding, testing, and debugging. The Xcode IDE combined with the Swift programming language make developing apps easier and more fun than ever

[Site web : Apple](#) ▶ [Assistance : Xcode](#) ▶ [Contrat de licence de l'app](#) ▶

...suite

Nouveautés de la version 8.3

Xcode 8.3 includes Swift 3.1 and SDKs for iOS 10.3, watchOS 3.2, tvOS 10.2, and macOS Sierra 10.12

- Siri support in the iOS Simulator to improve automated testing

Outils de développement

Xcode 8.2

Installer Xcode

- Pour l'installer il suffit de lancer le .dmg ou de l'installer directement à partir d' Apple Store.
- Garder Xcode, après lancement, sur le code grâce à
Xcode -> Options -> Garder dur le Dock

Outils de développement ^{1/2}

Premier programme avec Xcode

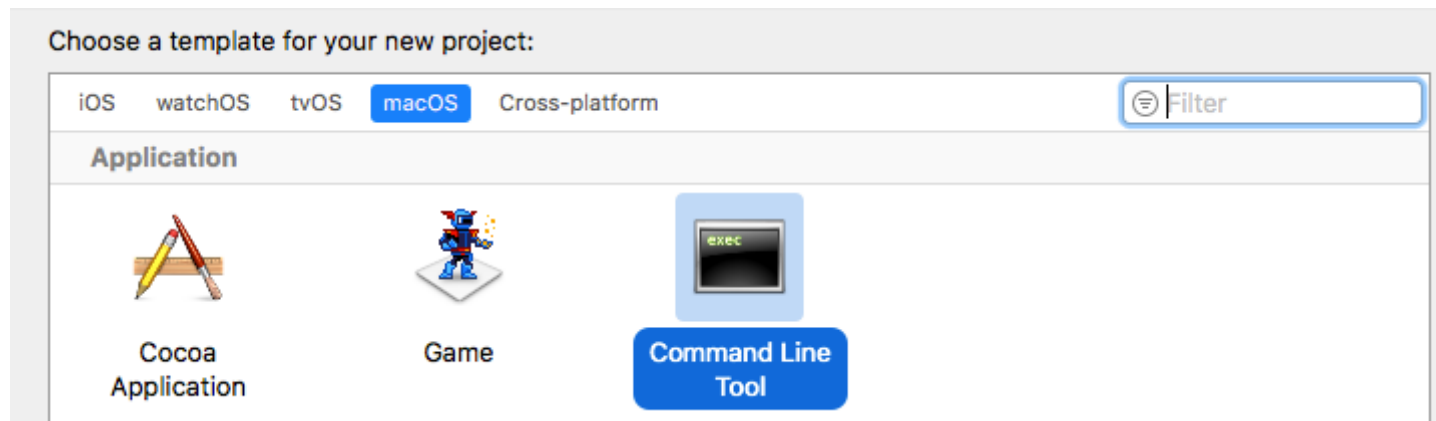
Pour créer un nouveau projet:



Create a new Xcode project

Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.

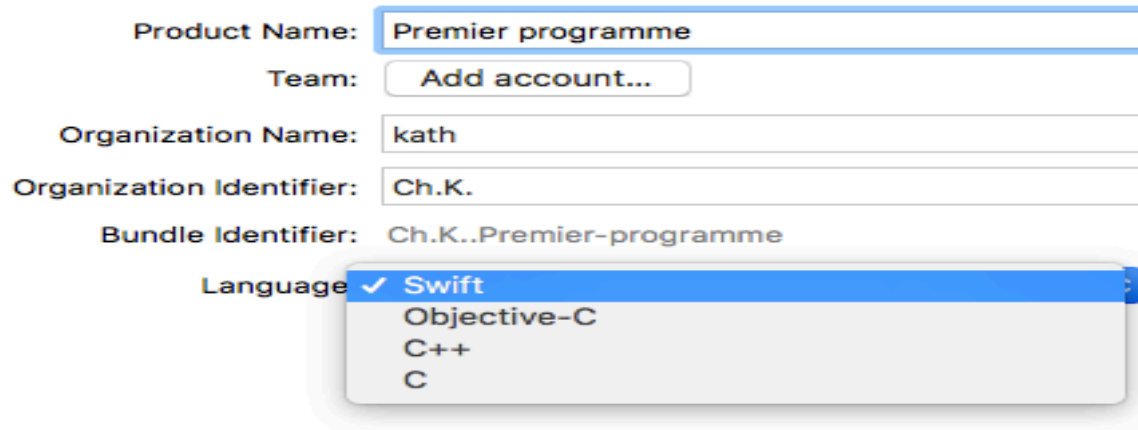
Pour acquérir les bases du langage on commence par *Commande Line Tool* sous *macOS*



Outils de développement 2/2

Premier programme avec Xcode

Nommer le projet et choisir le langage de programmation Swift :



Product Name: Premier programme

Team: Add account...

Organization Name: kath

Organization Identifier: Ch.K.

Bundle Identifier: Ch.K..Premier-programme

Language: ☒ Swift
Objective-C
C++
C

Les bases du langage Swift

Les variables et opérations

Déclaration de variable :

En swift la déclaration d'une variable est précédée du mot clé *var* .

- Déclaration implicite : Sans préciser le type de données

var variable = 2

- Déclaration explicite : En précisant le type de données dans notre exemple Int

var variable : Int = 2

Les variables et opérations

Les différents types :

Int : integer, entier

Float: float, nom à virgule sur 4 octets

Double: double, nombre à virgule sur 8 octets

Bool: boolean, valeurs (*true, false*)

String: Chaîne de caractères

Character: caractère (char en C)

Les variables et opérations

Déclaration de constantes :

La déclaration des constantes est quasi identique aux variables **sauf que** le mot clé utilisé est *let* à la place de *var*

Contrairement aux variables qui peuvent être modifiées (muables) les constantes sont immuables.

- Déclaration implicite : Sans préciser le type de données
let constante = 2
- Déclaration explicite : En précisant le type de données
dans notre exemple Int
let constante: Int = 2

Afficher le contenu

- Pour afficher le contenu d'une variable ou une constante en utilisant la fonction prédéfinie

Print()

Exemple :

```
let a = 2
```

```
print("le contenu de",a,"donc a est un entier")
```

```
let chaine = "bonjour"
```

```
print("le contenu de ma variable est "+chaine+"!!")
```

Afficher le contenu 1/2

- Interpolation grâce à *Print()*

Exemple :

```
let a = 2
```

```
print("le contenu de \a) donc a est un entier")
```

```
let chaine = "bonjour"
```

```
print("le contenu de ma variable est \chaine!!!")
```


Importe / commentaires ^{1/2}

- Commentaires :

// : Commentaire sur une seule ligne

/* */ : Commentaire sur plusieurs lignes

- Fichier import Foundation

import Foundation

– Au début du fichier **main.Swift**

– Permet d'accéder à NSObject et ses sous-classes

Importe / commentaires 2/2

- **Fichier import Foundation (suite)**
- Accéder aux classes essentielles qui définissent le comportement des objets de base, les types de données et services du système d'exploitation.
- Incorporer les modèles de conception et des mécanismes qui rendent vos applications plus efficaces et robustes.

Exercice pratique 1

- Lancez et créez un projet nommé ExercicesCours
- Ecrire un petit programme dans main.swift qui déclare, initialise et affiche le contenu des différents types vus précédemment.

Les tuples ^{1/2}

- Les tuples sont un type de variables contenant plusieurs valeurs.
- La syntaxe
let tuple (valeur1, valeur2, valeur3, ...)
Ou *var* tuple (valeur1, valeur2, valeur3, ...)

Les tuples 2/2

Exemple

```
//déclarer un tuple perosonne de deux valeurs
var personne = (prenom:"", nom: "")
//nitilaiser les tuples
personne.nom="toto"
personne.prenom="Dupond"
//afficher le contenu d'un tuple
print("le nom et prénom de cette personne :  \(personne.nom)  \(personne.prenom) ")
// affecter les valeurs d'un tuple à un autre tuple
var (valeur1,valeur2) = personne
//afficher le contenu
print("le nom et prénom de cette personne :  \(valeur1) et \(valeur2) ")
```

Les conditions

Programmes séquentiels ne résolvent pas tous les problèmes

exemple simple : calcul des racines d'un polynôme de d° 2 dans \mathbb{R}

Algorithme pour la résolution de $a.x^2+b.x+c=0$

calculer $\Delta=b^2-4ac$

si $\Delta < 0$: pas de solution

si $\Delta \geq 0$: solutions : $x_1 = \frac{-b + \sqrt{\Delta}}{2a}$ et $x_2 = \frac{-b - \sqrt{\Delta}}{2a}$

Test sur la valeur de Δ

Les conditions

autre exemple simple : saisie en contrôlant la valeur d'entrée

une valeur à saisir doit absolument être supérieure à une borne b_inf .

algorithme :

saisir la valeur

tant que la valeur est $< b_inf$, recommencer la saisie.

Réaliser un test : conditions logiques

if sert à réaliser un test selon une **condition logique**

cette condition sera évaluée et aura une valeur de vérité : ***vraie*** ou ***fausse***. C'est selon cette valeur, vraie ou fausse, que l'on peut choisir de faire certaines instructions ou non.

Utilisation d'opérateurs de comparaison :

2 valeurs sont elles-égales ? \Rightarrow réponse de type vrai ou faux

1 valeur est-elle supérieure (au sens strict ou au sens large) à une autre ? \Rightarrow réponse de type vrai ou faux

Réaliser un test : conditions logiques

Opérateurs de comparaison :

Syntaxe	emploi	vrai si
==	val1 == val2	val1 est égal à val2
!=	val1 != val2	val1 est différent de val2
<	val1 < val2	val1 strictement inférieur à val2
<=	val1 <= val2	val1 inférieur ou égal à val2
>	val1 > val2	val1 strictement supérieur à val2
>=	val1 >= val2	val1 supérieur ou égal à val2

Attention à l'opérateur d'égalité ==, il ne faut pas le confondre avec l'opérateur d'affectation =

Opérateurs logiques

En fait, une condition est une expression qui donne une valeur de type vrai ou faux.

Il existe des opérateurs logiques permettant de construire des expressions logiques (de même que l'on construit des expressions mathématiques avec des opérateurs mathématiques).

Arithmétique des valeurs logiques :

opérateurs ET, OU et NON

ET et OU : composent 2 valeurs logiques

NON : s'applique à une valeur logique

Opérateurs logiques

Tables de vérité des opérateurs logiques. Soient A et B deux conditions, qui peuvent prendre les valeurs VRAI ou FAUX.

- L'expression A ET B a pour valeur VRAI ssi A et B ont pour valeur VRAI en même temps, sinon A ET B vaut FAUX.
- L'expression A OU B a pour valeur VRAI si A est VRAI ou si B est VRAI.
- L'expression NON A a pour valeur VRAI si A a pour valeur FAUX, et vaut FAUX si A a pour valeur VRAI.

Traduction en C :

ET : &&

OU: || (2 barres verticales, Shift+alt+L)

NON : !

Opérateurs logiques

Remarque sur les parenthèses des expressions logiques :

&& est prioritaire sur ||.

Avec Swift les parenthèses ne sont pas obligatoires mais si vous définissez une priorité, il faut systématiquement utiliser des parenthèses (les plus prioritaires) pour construire les expressions.

Syntaxe de l'instruction **if**

Réaliser un test simple : si une condition est vraie alors faire une instruction.

```
if condition
```

```
{
```

```
Instruction
```

```
} // Les accolades sont obligatoires même pour une  
seule instruction
```

ou

si la condition est fausse, l'instruction (ou le bloc) ne sera pas effectué.

l'instruction **if** dans un programme

Programme exemple : test d'une valeur

```
if u > b && u < c
{
    print("u appartient à l'intervalle ouvert u,c ")
}
```

Remarques:

- Les accolades sont obligatoires
- Les parenthèses ne sont pas obligatoires
- Bien espacer les opérateurs de comparaison des variables

L'alternative : **if ... else**

Syntaxe :

```
if condition
{
    instructions; /* si la condition est vraie */
}
else
{
    autres instructions; /* si elle est fausse */
}
```

L'alternative dans un programme

```
if mystere != solution
{
    printf("Révissez vos classiques !\n");
}
else /* si la condition est fausse */
{
    printf("Bravo !\n");
}
}
```


Conditions et alternatives

Avec une condition de type

`val1 == val2`

`val1 != val2`

`val1 < val2`

`val1 <= val2`

de même pour `>` et `>=`

Les instructions du bloc lié au else seront faites si

`val1 différent de val2`

`val1 est égal à val2`

`val1 supérieur ou égal à val2`

`val1 strictement supérieur à val2`

Instruction de sélection (Switch)

Dans le cas où nous avons énormément de conditions imbriquées, il est possible de simplifier en utilisant un switch:

La syntaxe :

```
switch choix
{
    case 1 : //Le choix saisi est 1
    case 2,3,4 ://Le choix est soit 2 ou 3 ou 4
    case 5...10 ://Le choix va de 5 à 10 inclus
    case 5...<10 ://Le choix va de 5 à 9 inclus
    default : //choix par défaut
}
```

Exercice appliqué 2

if...else permet de traiter deux conditions exclusives l'une de l'autre.
Mais pour 3 conditions exclusives ? Ou plus ?

Exemple : Résolvez une équation de d° 2 dans \mathbb{R} , en tenant compte du cas $\Delta=0$: 3 cas à traiter (*initialisez les variables*)

si $\Delta < 0$: pas de solutions

si $\Delta = 0$: une racine réelle double

si $\Delta > 0$: deux racines réelles distinctes

3 conditions exclusives :

Δ est forcément strictement négatif ou strictement positif ou nul !

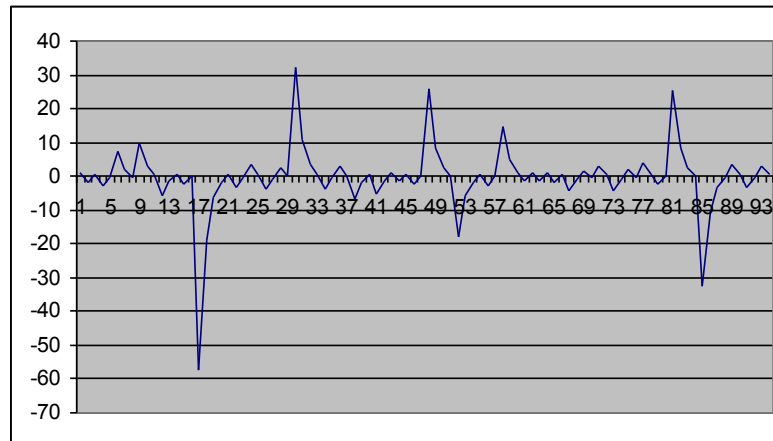
Les boucles / Itérations

Opération à répéter un certain nombre de fois :

exemple des suites mathématiques :

$$\left. \begin{array}{l} U_0 \text{ donné,} \\ U_{n+1} = f(U_n) \end{array} \right\}$$

exemple : $U_0 = 1, U_{n+1} = \frac{1}{3} \cdot U_n - \frac{2}{U_n}$



Les itérations

On veut la valeur de U6 : calcul par suite d'instructions simples

illustration :

```
var terme_U = 1.0    //initialisation du U avec U0
terme_U = (terme_U/3.0)-(2.0/terme_U) // calcul U1
terme_U = (terme_U/3.0)-(2.0/terme_U) // calcul U2
terme_U = (terme_U/3.0)-(2.0/terme_U) // calcul U3
terme_U = (terme_U/3.0)-(2.0/terme_U) // calcul U4
terme_U = (terme_U/3.0)-(2.0/terme_U) // calcul U5
terme_U = (terme_U/3.0)-(2.0/terme_U) // calcul U6

print("U6 = \n",terme_U)
```

Les itérations

6 fois la même instruction !

Si l'on veut calculer U_{150} , on devra répéter cette instruction 150 fois !

Réécrire le programme pour chaque valeur différente !

Autre exemple : l'ordinateur choisit un chiffre, l'utilisateur doit le deviner en proposant des valeurs. A chaque proposition, l'ordinateur indique si la proposition faite est supérieure, inférieure ou égale.

Les itérations

Avec des if .. else on ne fait les tests qu'une fois ! Il faudrait faire ces tests jusqu'à ce qu'on trouve effectivement le bon résultat !

Il existe en Swift des instructions permettant des itérations : répétition d'instructions quand une condition est satisfaite :

while() : tant que

repeat... while() : faire tant que

syntaxe :

while condition {

instruction

}

Les itérations

Avec **repeat...while**, on fait au moins une fois les instructions du corps de la boucle

Avec **while**, on peut ne pas faire les instructions de la boucle, si la condition est fausse dès le premier test !.

Exemple d'applications : saisie sécurisée d'une valeur.

Pour tester si une valeur est correcte, il fait d'abord la saisir !

Si cette valeur est incorrecte, on la ressaisira : on utilise alors une boucle **repeat...while**, puisqu'il faut faire les instructions (ici la saisie) AVANT de faire le test de la condition !

Répétition avec la boucle for ^{1/2}

Dernier type de boucle : la boucle for, qui sert essentiellement à répéter une ou des instructions un nombre de fois déterminé.

Conceptuellement très proche de la boucle while.

En incrémentant de 1 :

Exemple :

```
for _ in 1...5 {  
    print("Hello World")  
}
```

Répétition avec la boucle for

En incrémentant avec d'autre pas > 1 :

Exemple : en utilisant un pas de 2 :

```
for i in stride(from: 0, to: 10, by: 2)
{
    print("i= ", i)
}
```

Exercice Appliqué 5

Ecrire un programme qui permet d'afficher les n nombres premiers par pas de 3 en utilisant les différentes boucles vues précédemment