Table des matières

[Comprendre l’écosystème iOS 3](#_Toc476242438)

[Pourquoi iOS ? 3](#_Toc476242439)

[Fragmentation 4](#_Toc476242440)

[Taux d’adoption 4](#_Toc476242441)

[Pourquoi faire du Natif ? 5](#_Toc476242442)

[Applications Web 5](#_Toc476242443)

[Librairies mobiles multiplateformes 5](#_Toc476242444)

[SDK iOS 6](#_Toc476242445)

[L’écosystème iOS 6](#_Toc476242446)

[Appareils 6](#_Toc476242447)

[iOS 7](#_Toc476242448)

[App Store 7](#_Toc476242449)

[watchOS et tvOS 7](#_Toc476242450)

[Configuration de l'environnement de développement 8](#_Toc476242451)

[Exigences et conditions préalables 8](#_Toc476242452)

[Adhérer au programme Apple Développeur 8](#_Toc476242453)

[Création d'un Apple ID 9](#_Toc476242454)

[Inscription à l'Apple Developer Program 9](#_Toc476242455)

[Installation d'Xcode 10](#_Toc476242456)

[Création d'une application 11](#_Toc476242457)

[L'interface utilisateur d'Xcode 13](#_Toc476242458)

[La barre d'outils 13](#_Toc476242459)

[Navigateur 14](#_Toc476242460)

[Vue principale ou espace de travail 14](#_Toc476242461)

[Inspecteur 14](#_Toc476242462)

[Build & Run 14](#_Toc476242463)

[Le Simulateur 14](#_Toc476242464)

[Périphérique physique 15](#_Toc476242465)

[Le langage de programmation 16](#_Toc476242466)

[Origines de Swift 16](#_Toc476242467)

[Playgrounds 17](#_Toc476242468)

[Variables et constantes 18](#_Toc476242469)

[Types de données 18](#_Toc476242470)

[Booléens 18](#_Toc476242471)

[Numbers 18](#_Toc476242472)

[Strings 19](#_Toc476242473)

[Collections 19](#_Toc476242474)

[Any et AnyObject 19](#_Toc476242475)

[Classes et Structures 20](#_Toc476242476)

[Classes 20](#_Toc476242477)

[Structures 20](#_Toc476242478)

[Classe vs Structure 20](#_Toc476242479)

[Optionals 21](#_Toc476242480)

[Qu'est-ce qu'un optional ? 22](#_Toc476242481)

[Forced unwrapping 22](#_Toc476242482)

[Optional Binding 22](#_Toc476242483)

[Qu'est-ce que nil ? 23](#_Toc476242484)

[Où est passé le point-virgule ? 23](#_Toc476242485)

[Fonctions 23](#_Toc476242486)

[Qu'est-ce qu'une fonction ? 23](#_Toc476242487)

[Paramètres 23](#_Toc476242488)

[Types de retour 24](#_Toc476242489)

[Closures 24](#_Toc476242490)

[Qu'est-ce qu'une closure ? 25](#_Toc476242491)

[Syntaxe 25](#_Toc476242492)

[Closures en paramètres 25](#_Toc476242493)

[Trailing closures 25](#_Toc476242494)

[Protocols 26](#_Toc476242495)

[Contrôle d'accès 27](#_Toc476242496)

[Créer votre première application 28](#_Toc476242497)

[Exploration du projet 28](#_Toc476242498)

[Modifier l'interface utilisateur 28](#_Toc476242499)

# Comprendre l’écosystème iOS

Avant de commencer à concevoir des applications iOS, il est important de comprendre son écosystème. Cette partie aborde les fondamentaux du système d’exploitation d’Apple, les périphériques disponibles et l’App Store d’Apple.

## Pourquoi iOS ?

Avant l’arrivée d’Android et de Windows Phone, Apple était l’acteur principal de l’univers de mobile, seuls Nokia et BlackBerry rivalisaient. Avec l’émergence de nouvelles plates-formes mobiles, il est devenu de plus en plus important de se demander quelle est celle qui vous convient le mieux.

**En dépit de l’avance d’Apple en 2007, Android a dépassé iOS en parts de marché. Qu’es ce que cela signifie pour iOS ? S’agit-il encore d’une plates-forme pour laquelle il faut développer ? Bien sur ! Il existe de nombreuses bonnes raisons de commencer à développer iOS aujourd'hui.**

* **Financière**  
  Les gens dépensent plus d'argent sur l'App Store d'Apple que sur toute autre plate-forme mobile. Malgré le fait qu'Android est devenu l’acteur dominant dans l'espace mobile, la plate-forme iOS reste plus lucrative pour les développeurs.
* **Espace fermé**  
  Au fil des ans l’App Store d'Apple a reçu beaucoup de critiques des développeurs et des clients notamment sur le processus d'examen. Mais il est clair, que la politique d'Apple a conduit à un store dans lequel la qualité de ses applications est plus élevée que dans tout autre magasin mobile. Les clients ont également tendance à se sentir beaucoup plus en sécurité lors de l'achat d'applications de l'App Store que sur d'autres plates-formes plus ouvertes, telles que Google Play.
* **Communauté et support**La communauté des développeurs iOS est très riche. Il y a presque une offre illimitée d’articles, de tutoriels, de documentations et des projets exemples pour vous aider lorsque vous vous avez un problème. En outre, la documentation d’Apple sur le SDK iOS est très complète, sans oublier les centaines de vidéos WWDC (conférence annuelle des développeurs) disponibles sur le portail développeurs d’Apple.

Les gens oublient souvent, ou ne savent pas, que Android et iOS sont des plates-formes très différentes. Même si les deux plates-formes ont été conçues pour l'espace mobile, l'écosystème iOS diffère à bien des égards de l'écosystème Android.

**Du point de vue du développement, la plupart des gens trouvent plus facile de développer pour iOS que pour Android. Cela ne se réfère pas à la pile de technologie que chaque plate-forme utilise ni aux outils disponibles pour chaque plate-forme. Deux raisons sont à la base de l'abîme entre iOS et Android, la fragmentation et le taux d'adoption.**

### Fragmentation

Une des principales forces d’Android est également une de ses principales faiblesses. Comme la plupart d'entre vous le savent, le système d'exploitation Android est publié en tant que projet open source et peut donc être **utilisé et modifié par quiconque**. De nombreux fabricants de matériel ont vu le potentiel d'Android, d'autant plus qu'il est soutenu par Google.

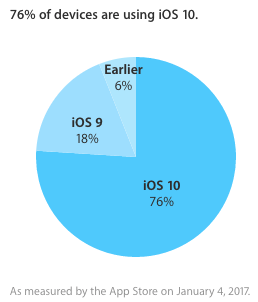
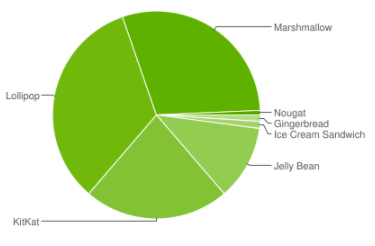
**Ainsi, Android a rapidement gagné en popularité. Le problème est que chaque fabricant de matériel modifie Android pour l'adapter à leurs besoins et une grande variété de déclinaisons ont émergé depuis l'introduction d'Android**.

Même si la part de marché d'Android peut sembler comme un grand pot de miel, c'est une peine de créer un logiciel qui fonctionne sur chaque version Android actuellement disponible. En raison des milliers de différents dispositifs qui exécutent Android, les tests de logiciels sur le matériel n'est pas une tâche banale. Comme vous pouvez l'imaginer, le support et la compatibilité deviennent un réel défi.

### Taux d’adoption

Traditionnellement, dans de nombreux pays, le marché mobile est sous contrôle strict par les opérateurs cellulaires. Dès le début, Apple savait qu'il avait besoin d'être en contrôle de ses appareils mobiles pour fournir la meilleure expérience possible. Le résultat est que les clients peuvent mettre à niveau leurs appareils iOS à la dernière version d'iOS le jour même où il est publié au public. C'est un avantage majeur de la plate-forme iOS et le résultat est substantiel. Moins d'une semaine après la sortie d'iOS 9, plus de 50% des appareils iOS avaient été mis à jour avec cette version. Actuellement 76% des terminaux iOS utilisent la dernière version du système d’exploitation iOS 10.

En comparaison avec Android ; en septembre 2015, seulement 20% des utilisateurs d'Android avaient mis à niveau leurs appareils vers Lollipop (sorti en 2014) et moins de 40% avaient mis à niveau vers KitKat (publié en octobre 2013).

La raison pour élever ces chiffres n'est pas de minimiser la plate-forme Android ou toute autre plate-forme mobile. Ce qu’il faut retenir, c'est l'impact du taux d'adoption sur l'écosystème d'application de ces plates-formes.

Les utilisateurs plus lents adoptent une nouvelle version d'un système d'exploitation, plus les développeurs sont forcés de prendre en charge les anciennes versions du système d'exploitation. Il devient également moins attrayant d'adopter de nouvelles technologies, qui ne peuvent être utilisées que sur une fraction des appareils Android.

## Pourquoi faire du Natif ?

Si vous êtes curieux et que vous avez déjà abordé le sujet du mobile, vous avez peut-être entendu parler de librairies multiplateformes comme Xamarin, React Native, PhoneGap, Apache Cordova ou encore Titanium Mobile. Vous pourriez vous demander pourquoi vous devriez passer par la difficulté d'apprendre un nouveau langage de programmation (Swift ou Objective-C) quand vous pouvez créer des applications iOS en utilisant C # ou JavaScript. Permettez-moi de vous donner un bref aperçu de vos options si vous décidez de développer des applications pour iOS.

### Applications Web

Une application Web est votre première option et c'était en fait la seule option avant l'introduction du SDK natif iOS en 2008. L'avantage des applications Web est qu'ils n'ont pas à traiter avec l'App Store ou l'examen d'Apple processus. Pourquoi donc ? Une application Web est un nom de fantaisie pour un site Web qui agit et se comporte comme une application. Tout ce dont vous avez besoin est un navigateur, tel que Safari ou Google Chrome.

Les applications Web présentent des avantages évidents. Ils sont multiplateformes par défaut puisque les applications s'exécutent dans un navigateur. Un autre avantage est la courbe d'apprentissage pour créer des applications Web. Si vous êtes familier avec le développement web, alors vous serez opérationnel en un rien de temps.

Bien sûr, les inconvénients des applications Web ne doivent pas non plus être ignorés. Une application Web a un accès limité aux fonctionnalités des périphériques sur lesquels ils s'exécutent. Les applications Web mobiles sont très puissantes grâce à HTML5 et JavaScript. Cependant, il existe des limites quant à ce qu'ils peuvent faire. Vous ne serez pas en mesure d'utiliser les fonctionnalités cool de l'iPad et l'iPhone. En outre, les applications Web nécessitent généralement une connexion réseau pour fonctionner.

Peut-être le plus grand inconvénient de la construction d'applications Web est que, en général, elles semblent beaucoup plus lentes que les applications natives. La réactivité des applications Web s'améliore chaque année, mais le retard hérité ajouté par le navigateur ne doit pas être négligé ou ignoré. L'expérience est différente. Cela ne fait aucun doute.

### Librairies mobiles multiplateformes

La promesse d'un framework multiplateforme est que les développeurs peuvent écrire une application qui peut être déployée sur plusieurs périphériques mobiles à partir d'une base de code unique. Ceci est réalisé par un certain nombre d'approches.

Certains frameworks, comme PhoneGap et Apache Cordova, utilisent le SDK natif de la plate-forme pour créer une vue Web dans laquelle l'application est intégrée. Comme pour les applications Web, cela impose des limites en termes de fonctionnalité. Le principal avantage est que l'application se sent comme une application native en termes d'expérience utilisateur et qu'il peut être distribué par le store de la plate-forme.

Même si ces librairies peuvent apparaitre comme le Saint Graal pour le développement mobile, il y a un certain nombre d'inconvénients qui doivent être pris en considération. L'inconvénient le plus important est que votre application dépend d'un cadre tiers. En d'autres termes, la base de code de votre projet dépend de l'entreprise qui fournit la solution multiplateforme. Si cette société quitte l'entreprise ou est lente à mettre en œuvre les modifications apportées au SDK natif, vous pourriez être forcé de recommencer à zéro. C'est un risque que vous devez prendre en considération.

### SDK iOS

Choisir de développer avec le SDK natif est le meilleur choix si vous voulez créer des applications qui se démarquent et profiter des capacités de l'appareil. Opter pour une application native signifie également que vous travaillerez dans un environnement de développement robuste et que vous pourrez compter sur les outils de développement, les utilitaires et le support d'Apple.

## L’écosystème iOS

L'un des aspects les plus attrayants de l'écosystème iOS est sa simplicité. La liste des périphériques capables d'exécuter iOS est étonnamment courte par rapport à Android ou Windows Phone. Cela signifie que le test des applications iOS est beaucoup moins encombrant par rapport aux autres plates-formes.

Comme mentionné précédemment, Android a dépassé iOS en termes de parts de marché, mais ce n'est qu'une partie de l'histoire. La plate-forme iOS représente plus d'un tiers de l'utilisation du Web sur les appareils mobiles.

Qu'est-ce que cela nous dit sur l'espace mobile ? Il nous dit que iOS reste un acteur dominant dans l'espace mobile, au moins en ligne, et cela en dépit de la part de marché beaucoup plus importante d'Android.

### Appareils

Il existe essentiellement deux familles de périphériques dans l'écosystème iOS :

* Celle des iPad
* Celle des iPhone, qui comprend également l’iPod Touch

Pour un aperçu complet, voici une [liste détaillée](http://blakespot.com/ios_device_specifications_grid.html)[[1]](#footnote-1) des périphériques. Les principales différences au sein de la famille d'appareils iPhone sont les capacités matérielles et la taille de l'écran.

Pour l’iPhone et l’iPod Touch, il y a 4 tailles d’écran possibles :

* 3.5" (iPhones < 4/4s et iPod Touch < 4G)
* 4.0" (iPhones 5/5s/5c/SE et iPod Touch 5G)
* 4.7" (iPhones 6/6s/7 et iPod Touch 6G)
* 5.5" (iPhones 6 Plus / 6s Plus / 7 Plus)

Pour la famille des iPads, il y a 3 tailles d'écrans possibles :

* 7.9" (iPad mini)
* 9.7" (iPad, iPad Air, iPad Pro)
* 12.9" (iPad Pro)

La différence de taille affecte le développement, mais il n'ajoute pas beaucoup de frais généraux pour la plupart des applications. Nous le verrons davantage dans la suite.

En 2012, Apple a présenté l'iPad Mini et en 2015, la famille iPad a été agrandie avec l'iPad Pro. Bien que les introductions de l'iPad Mini et de l'iPad Pro aient provoqué une vague dans l'industrie de la technologie, elle a eu peu d'impact pour les développeurs iOS.

L'écran de l'iPad Mini, par exemple, est nettement plus petit que celui de l'iPad Air 9.7", mais la résolution est identique à la résolution des appareils iPad 9.7". En raison de l'écran plus petit iPad Mini, la taille moyenne des cibles tactiles est plus petite et cela pourrait poser un problème pour certaines applications.

### iOS

Traditionnellement, Apple lance une nouvelle version de son système d'exploitation mobile chaque année. Jusqu'à présent, Apple a publié dix versions majeures de son système d'exploitation mobile avec la dernière version étant iOS 10.

Le support de périphérique est un aspect d'iOS que les gens oublient souvent. Prenez l'iPhone 4S, par exemple. L'iPhone 4S a été publié en 2011 et est toujours capable d'exécuter iOS 9. Il n'y a pas de plate-forme mobile qui s'approche de ce niveau de support.

### App Store

L'une des principales raisons de développer pour la plate-forme iOS est le succès incroyable de l'App Store. La vente de logiciels n'a jamais été aussi facile grâce à l'App Store d'Apple sur iOS et OS X. Avec plus d'un million d'applications, c'est devenu un peu encombré dans l'App Store pour iOS, mais il faut garder à l'esprit que le nombre de périphériques iOS continue également à croître exponentiellement.

L'espace mobile en est à ses balbutiements et il y a encore tant de potentiel qui attend d'être découvert. Comme avec le web à la fin des années 80, l'espace mobile continuera à croître à un rythme rapide et le montant d'argent qui peut être fait augmente avec chaque nouveau smartphone, smartwatch et tablette vendus.

## watchOS et tvOS

En 2015, Apple a annoncé deux nouveaux systèmes d'exploitation, watchOS pour Apple Watch et tvOS pour Apple TV. Apple a ouvert les deux plates-formes pour les développeurs tiers, apportant des applications tierces à votre poignet et salon.

Les outils pour le développement de watchOS et de tvOS sont identiques à ceux pour le développement d'iOS. C'est encore une autre raison de choisir l'écosystème iOS. Les systèmes d'exploitation d'Apple, macOS, iOS, watchOS et tvOS, sont étroitement intégrés et les possibilités sont virtuellement infinies.

# Configuration de l'environnement de développement

Avant de commencer à créer des applications iOS, il est nécessaire de configurer l'environnement de développement. Dans cette section, nous allons voir comment vous inscrire en tant que développeur Apple et installer les outils de développement dont vous avez besoin pour démarrer.

## Exigences et conditions préalables

Le principal objectif de ce chapitre est de commencer avec Xcode. Xcode est un remarquable environnement de développement intégré (IDE) développé par Apple. La grande majorité des développeurs OS X, iOS, watchOS et tvOS s'appuient sur Xcode pour la construction d'applications.

Xcode est uniquement compatible avec le système d'exploitation OS X d'Apple, il faut donc nécessairement un Mac pour exécuter cet IDE. Nous allons utiliser la version 8.2 d'Xcode (actuellement 8.2.1).

Cette version particulière de Xcode nécessite OS X 10.11.5 ou supérieur. Même s'il est possible de créer des applications iOS avec une version plus ancienne de Xcode, il est recommandé d'utiliser également Xcode 8.2 profiter des dernières fonctionnalités et éviter les problèmes de compréhension dans cette section.

## Adhérer au programme Apple Développeur

Les membres du programme développeur Apple peuvent créer et publier des applications pour OS X, iOS, watchOS et tvOS.

Si vous ne faites que tester (sur simulateur ou sur votre appareil) et que vous ne voulez prendre un abonnement tout de suite, vous pouvez continuer avec votre identifiant Apple. Depuis 2015, il est possible de développer des applications pour les plates-formes d'Apple avec rien de plus qu'un Apple ID. Tant que vous avez treize ans ou plus, n'importe qui peut créer un compte développeur Apple et utiliser Xcode.

Si vous envisagez de soumettre vos applications à l'App Store, vous devez vous inscrire au programme Apple développeur. Le programme de développeur Apple propose trois types d'inscription payantes : individuel, organisationnel et d'entreprise. Apple a également un programme dédié pour les établissements d'enseignement.

Les inscriptions individuelle et organisationnelle coûtent 99$ par année. Le programme d'entreprise coûte plus cher à 299$ par année. Le programme d'entreprise s'adresse aux structures qui souhaitent déployer des applications internes qui ne sont pas distribuées par l'App Store d'Apple.

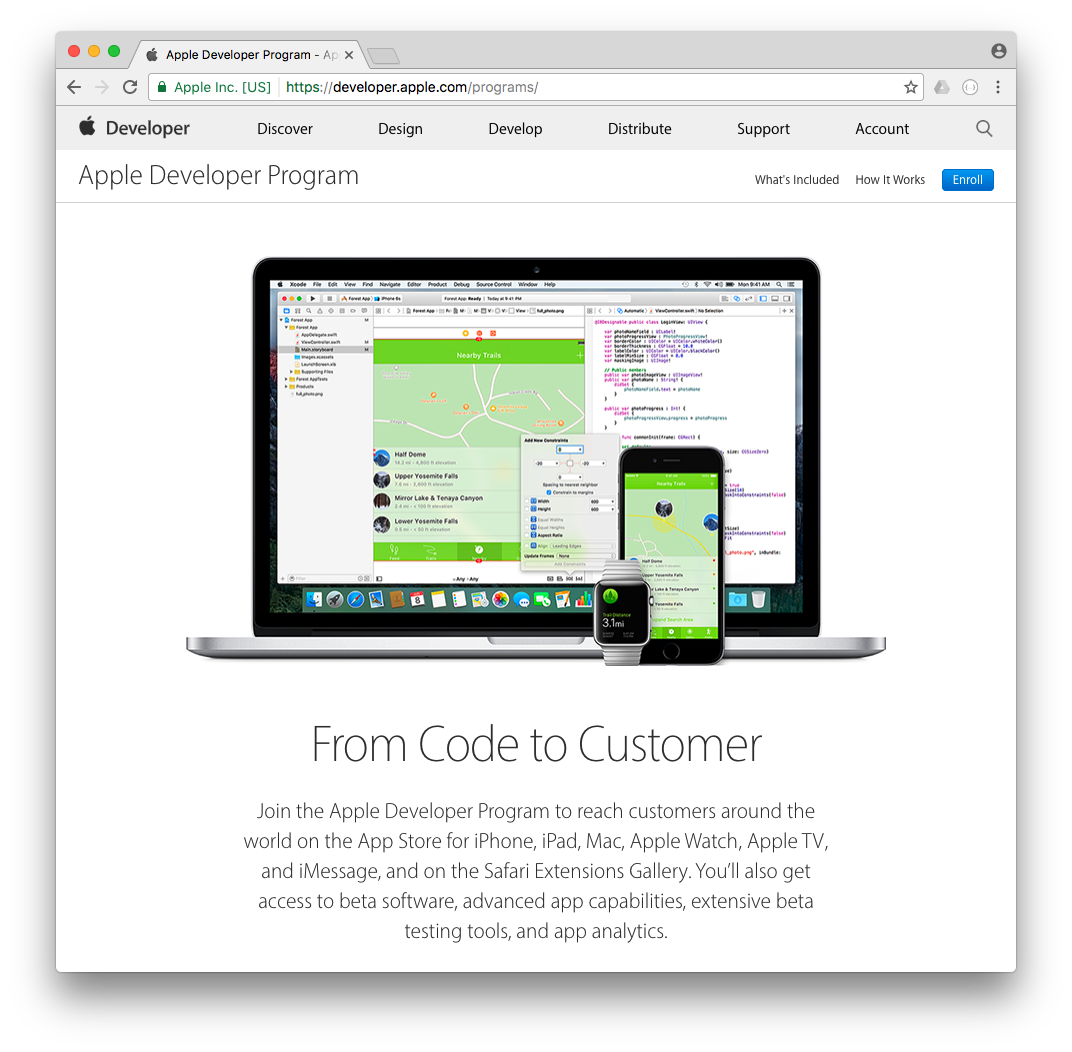
### Création d'un Apple ID

Avant de commencer à développer des applications iOS, vous avez besoin d'un identifiant Apple. Vous devez également vous inscrire au programme Apple Développeur. Vous pouvez en créer un gratuitement sur le site Web d'Apple. Il suffit de quelques minutes pour en créer un.

### Inscription à l'Apple Developer Program

Vous pouvez compléter cette série sans vous inscrire au programme Apple Developer, mais gardez à l'esprit que vous ne pourrez pas soumettre d'applications à l'App Store. Si vous vous inscrivez au programme développeur Apple, vous pouvez développer pour OS X, iOS, watchOS et tvOS. Visitez le site Web de développeurs d'Apple pour plus d'informations sur le programme développeur Apple.

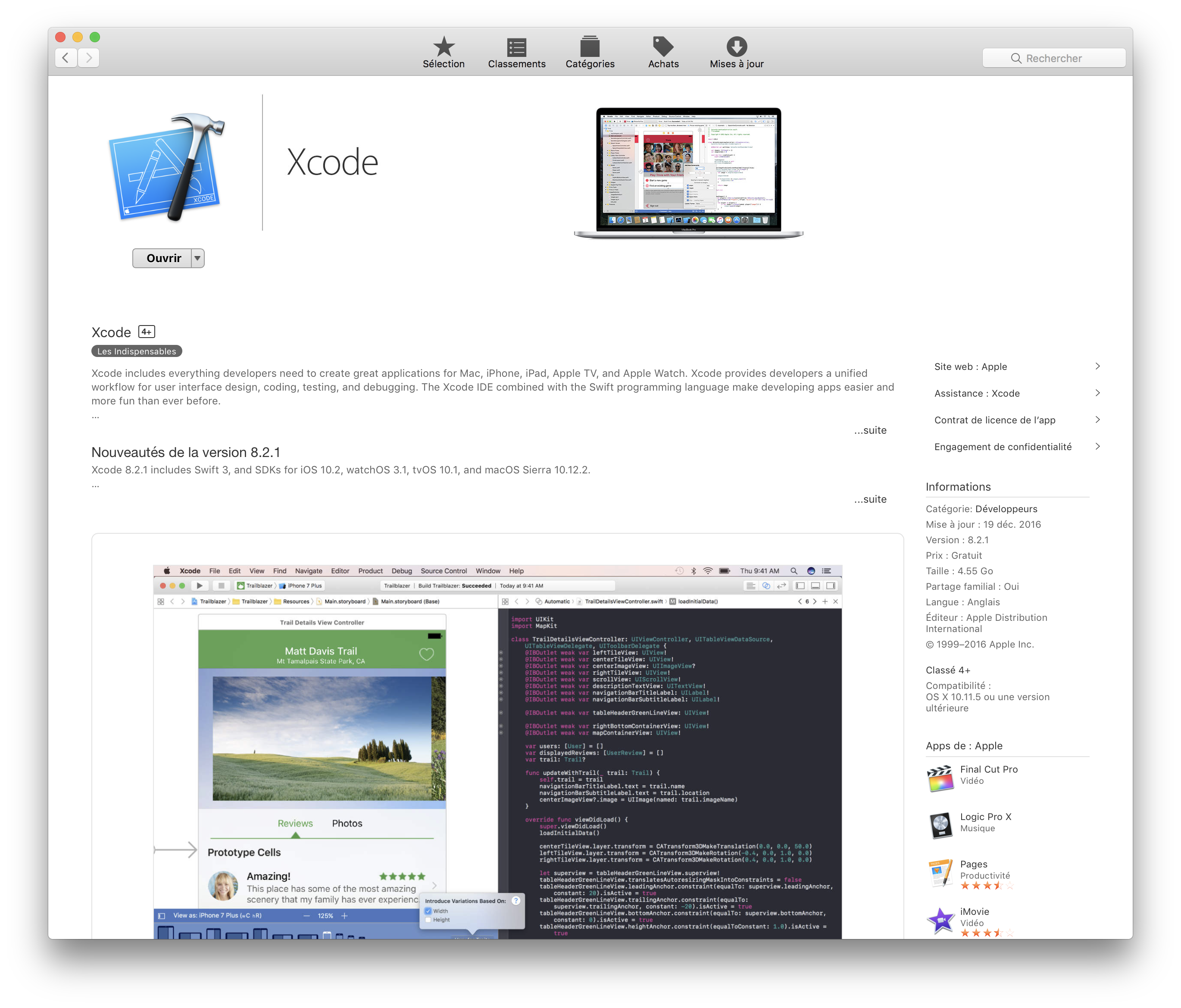
Si vous avez décidé de vous inscrire au programme de développement d'Apple, dirigez-vous ensuite vers le site Web du programme de développement Apple et cliquez sur Inscription en haut à droite. Connectez-vous avec votre identifiant Apple et suivez les étapes. Le processus peut prendre quelques jours pour compléter puisque Apple approuve manuellement chaque application. Pour plus d'informations, visitez le centre de support aux développeurs d'Apple.



## Installation d'Xcode

Le nom Xcode peut être un peu déroutant au début. Le nom est souvent utilisé pour désigner l'ensemble des outils macOS, iOS, watchOS et développement de TVOS, et ce jeu d'outils comprend l'IDE Xcode, le simulateur pour les applications en cours d'exécution et les OSK actuels OS X, iOS, watchOS et tvOS (Kit de développement logiciel). Cependant, il est important de comprendre que l'application Xcode elle-même est juste un IDE et quand j'utilise le nom Xcode, je me réfère généralement à juste cela.

Le téléchargement de Xcode peut s'effectuer de deux façons, via le site Web des développeurs d'Apple ou via l'App Store sur macOS. L'avantage de l'App Store est que la mise à jour de Xcode est plus facile et plus rapide.



L'avantage de télécharger Xcode via le site de développeurs est que vous pouvez télécharger des aperçus pour les développeurs pour tester les futures fonctionnalités en avance. Il est conseillé de télécharger et d'installer Xcode via l'App Store afin de travailler avec la version stable la plus récente.

Ouvrez l'App Store, recherchez Xcode et démarrez le processus d'installation. Attention, Xcode pèse plusieurs giga-octets.

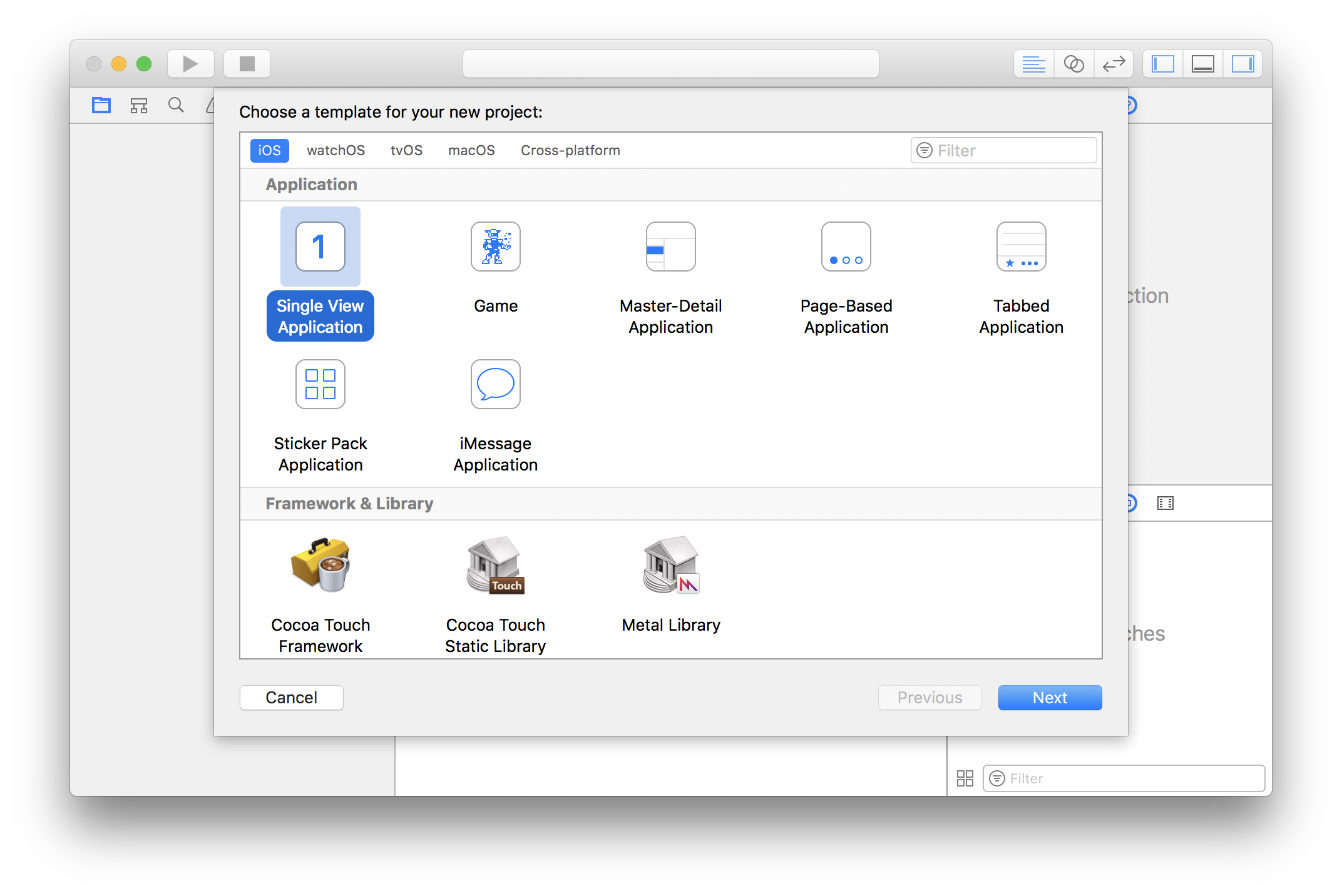
## Création d'une application

Avec Xcode installé, il est temps de le lancer pour la première fois. Si tout se passe bien, vous devriez voir la fenêtre **Welcome to Xcode**, qui contient quelques liens utiles et vous aide à créer une nouvelle application. Pour créer votre première application iOS, sélectionnez **Create a new Xcode project**.



Qu'est-ce qu'un projet Xcode ? Un projet Xcode est un dossier ou un package qui contient tous les fichiers et composants nécessaires à la gestion et à la construction de l'application.

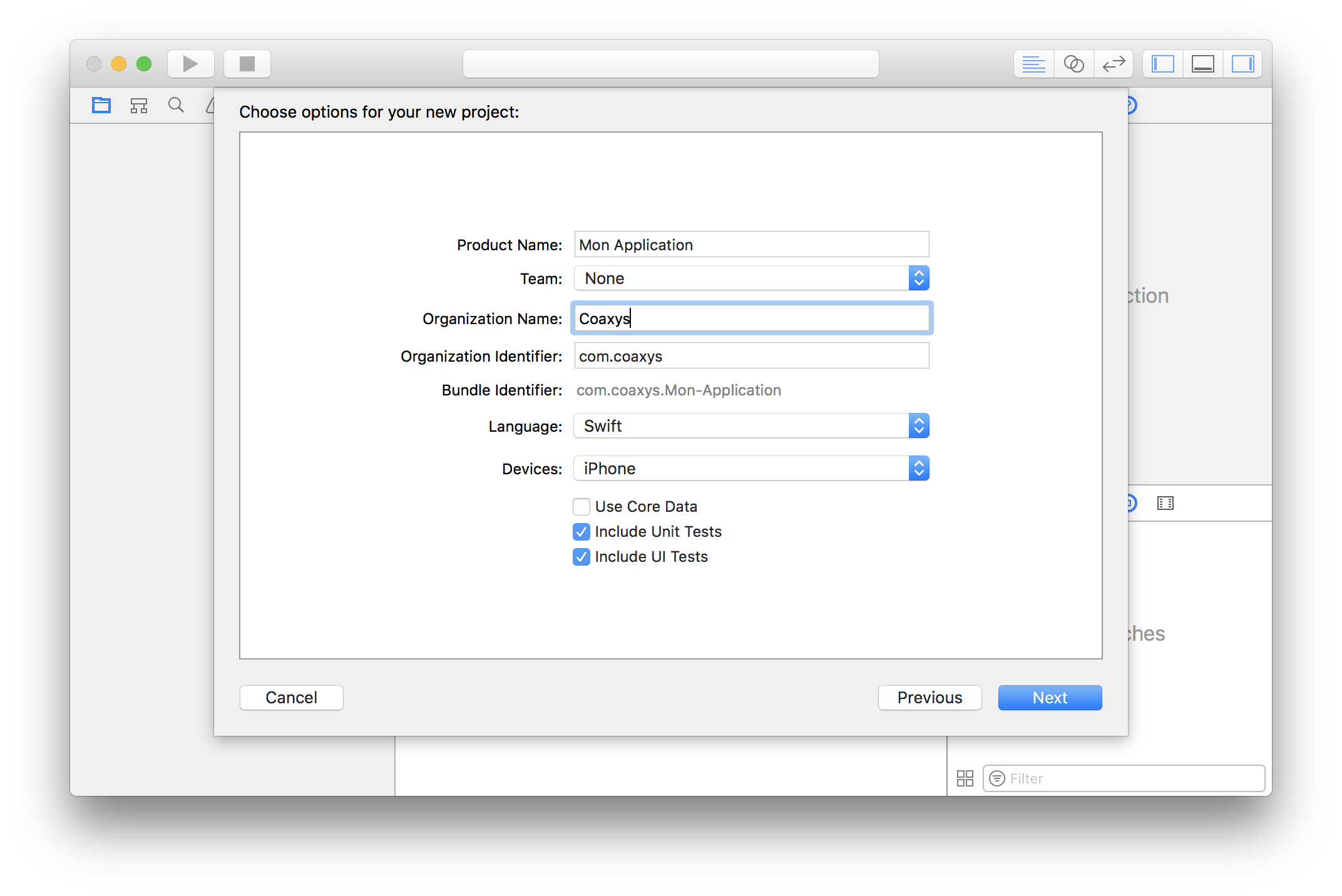
Xcode facilite la création d'un nouveau projet Xcode en proposant quelques modèles de projets utiles. La plupart du temps, le modèle **Single View Application** sera sélectionné. Sélectionnez **iOS** dans le bandeau supérieur et cliquez sur **Next**.



La fenêtre suivante vous permet de configurer votre projet Xcode au travers de multiples options :

* **Product Name**: le nom de votre application
* **Team**: L'Apple ID avec lequel vous allez signer votre application. Il peut être renseigné et changé par la suite.
* **Organization Name**: Le nom de l'organisation peut être votre propre nom ou le nom de votre entreprise. Xcode utilise le nom de l'organisation à diverses fins, comme l'ajout d'un avis de copyright à chaque fichier source.
* **Organization Identifier**: Il s'agit d'une chaîne unique utilisée par Xcode (avec le nom du produit) pour créer l'identificateur d'ensemble de l'application. Apple recommande d'adopter la notation de nom de domaine inverse pour éviter les collisions de dénomination.   
  Notez que le domaine que vous utilisez n'a aucun lien avec DNS. Par exemple, ma société s'appelle Coaxys et est située en France, ce qui signifie que mon identifiant de société est fr.coaxys. Cependant, je pourrais tout aussi bien utiliser com.coaxys ou fr.coaxys-toto.
* **Bundle Identifier**: Même si vous ne pouvez pas spécifier l'identificateur de lot lors de la création d'un nouveau projet Xcode, vous pouvez le modifier une fois que vous avez créé votre projet. Par défaut, l'identificateur d'ensemble est la combinaison de l'identificateur d'organisation et du nom du produit. Gardez à l'esprit que les espaces dans le nom du produit sont remplacés par des tirets dans l'identificateur de lot, car l'identificateur de lot ne peut pas contenir d'espace blanc.
* **Language**: La version actuelle de Xcode prend en charge à la fois Objective-C et Swift. Cette formation est axée sur le développement iOS en utilisant Swift 3.
* **Devices**: Cette option de configuration indique à Xcode quels périphériques cible votre projet. En sélectionnant la première option, Universal, votre projet cible les familles d'appareils iPad et iPhone. L'iPod Touch est un membre de la famille d'appareils iPhone comme vous l'auriez pu deviner.
* **Use Core Data**: Si vous cochez cette case, Xcode créera quelques fichiers supplémentaires et ajoutera du code pour vous aider à démarrer avec Core Data, le framework de persistance d'Apple pour OS X et iOS.
* **Include Unit Tests**: Les tests sont un aspect important du développement de logiciels. Xcode dispose d'un support intégré pour les tests unitaires.   
  En cochant cette case, XCode va ajouter un répertoire ou nos tests pourront être réalisés.
* **Include UI Tests**: Ajoute un répertoire et les librairies nécessaire aux tests de l'interface utilisateur.

Remplissez les champs comme indiqué dans la capture d'écran ci-dessous et cliquez sur **Next**. Dans un prochain article de cette série, je vais expliquer chacune des options de configuration plus en détail. L'objectif de cet article est de mettre votre première application en service dans le simulateur.



Dans la dernière étape, Xcode demande de spécifier un emplacement pour stocker votre nouveau projet Xcode. Il n'a pas vraiment d'importance où vous enregistrez le projet aussi longtemps que vous pouvez le trouver plus tard. Vous remarquerez également que Xcode offre la possibilité de créer un référentiel Git local pour votre projet. Il est fortement recommandé d'utiliser ce type de gestionnaire de versions de sources pour tout type de développement. Git est un excellent choix et c'est le système SCM (Source Control Management) le plus populaire dans la communauté macOS et iOS.

Il est important de se familiariser avec la gestion des sources, il est indispensable dans le développement de logiciels pour diverses raisons. Pour ne pas ajouter une couche supplémentaire de complexité, Git ne sera pas abordé dans la suite de ce document.

## L'interface utilisateur d'Xcode

Avant d'aller plus loin, il est nécessaire d'introduire les différentes parties de l'interface d'XCode. Il existe cinq espaces distincts : une barre d'outils en haut, des barres latérales droite et gauche, une zone de débogage en bas et la vue principale au centre.

### La barre d'outils

La barre d'outils en haut contient les boutons et les menus que vous utiliserez le plus souvent. Comme nous l'avons vu plus tôt dans cette série, les boutons pour exécuter et arrêter une application vivent également dans la barre d'outils.

L'affichage au centre de la barre d'outils est similaire à l'affichage que vous trouvez dans iTunes. Il affiche des informations sur l'état de votre projet (errors & warnings) ou sur ce que Xcode fait actuellement en arrière-plan. Il vous indiquera, par exemple, quand une compilation réussit ou échoue.

Les deux contrôles segmentés à droite de la barre d'outils peuvent être utilisés pour personnaliser l'interface utilisateur de Xcode. Jouez avec les différents contrôles pour savoir comment chacun d'eux change l'interface utilisateur de Xcode.

### Navigateur

Le but principal de la barre latérale gauche est pour la navigation et il est souvent appelé navigateur Xcode.

Le navigateur dispose de différents onglets avec le navigateur de projet à l'extrême gauche. La sélection ce navigateur détermine ce qui est affiché dans la vue principale de Xcode, l'espace de travail.

### Vue principale ou espace de travail

La vue principale ou l'espace de travail est la zone où vous passerez la plupart de votre temps. C'est le cheval de bataille de Xcode et affiche tout ce qui est sélectionné dans le navigateur.

### Inspecteur

Alors que la barre latérale gauche contrôle ce qui est affiché dans la vue principale Xcode, le contenu de la barre latérale droite reflète ce qui est affiché ou sélectionné dans la vue principale.

## Build & Run

Vous avez réussi à configurer votre premier projet Xcode. L'exécution de votre application dans le simulateur se lance en seulement un clic. Il y a un grand bouton de lecture à l'extrême gauche de la barre d'outils Xcode.

Avant d'exécuter votre application, assurez-vous que le schéma actif est défini sur l'iPhone 6 ou l'un des autres simulateurs iOS. Nous aborderons plus en détail les schémas plus loin dans la série.

Cliquez sur le bouton de lecture en haut à gauche pour compiler votre projet et exécuter votre application dans le simulateur. Vous pouvez également appuyer sur la touche Commande + R. Si tout s'est bien passé, vous devriez voir le simulateur exécuter votre application. Bien sûr, votre application ne fait rien d'utile encore. Nous réglerons cela plus tard dans cette série.

## Le Simulateur

Le simulateur est un outil précieux dans votre arsenal. Après le premier lancement, la compilation et l'exécution d'une application iOS dans le simulateur est plus rapide que le déploiement d'une application sur un périphérique physique. Il y a cependant une mise en garde. Pour diverses raisons, le simulateur ne reproduit pas parfaitement un appareil iOS.

Par exemple, le simulateur n'a pas de caméra ou d'accéléromètre, ses capacités GPS sont limitées à une liste d'emplacements et d'itinéraires prédéfinis et l'interaction de l'utilisateur est limitée aux gestes qui nécessitent un ou deux doigts. Aussi, gardez à l'esprit que le simulateur tire la puissance et les ressources de la machine sur laquelle il fonctionne. Cela signifie que le simulateur n'est pas adapté pour tester les performances des applications.

Le point est que vous devez toujours tester une application sur un périphérique physique avant de le soumettre à l'App Store ou de le distribuer aux testeurs. Même si vous n'avez qu'un seul appareil à tester, il est essentiel de tester vos applications sur un périphérique physique au lieu de compter uniquement sur le simulateur.

Malgré ses défauts, le simulateur est un outil très performant. Sélectionnez le simulateur et jeter un œil à son menu et les différentes options qu'il fournit. Le simulateur est parfaitement adapté pour tester les interfaces utilisateur sur différents facteurs de forme, imitant des gestes simples, simulant des avertissements en mémoire ou prenant des captures d'écran pour l'App Store. Vous y trouverez un outil très utile pour le développement d'iOS.

## Périphérique physique

À partir de 2015, l'exécution d'une application sur un périphérique physique est possible pour chaque développeur Apple, même si vous ne vous inscrivez pas dans le programme de développeurs Apple payé. Cela dit, l'exécution d'une application sur un périphérique physique implique quelques étapes supplémentaires. Ces étapes comprennent la configuration de votre projet et la configuration de l'appareil que vous prévoyez de tester. Parce que c'est un sujet assez complexe, nous ne l'aborderons pas dans la suite de cette formation.

# Le langage de programmation

## Origines de Swift

Dans la Worldwide Developer Conference 2014, Apple a surpris tous les développeurs iOS en lançant un nouveau langage de programmation appelé Swift. Swift est annoncé comme un langage de programmation "rapide, moderne, sûr, interactif". La langue est plus facile à apprendre et est livré avec des fonctionnalités pour rendre la programmation plus productive.

Avant l'annonce de Swift, les applications iOS étaient principalement écrites dans Objective-C. Le langage existe depuis plus de 20 ans et a été choisi par Apple comme le premier langage de programmation pour Mac et iOS. Pour la plupart des développeurs iOS, Objective-C était difficile à apprendre et que sa syntaxe semblait bizarre. Autrement dit, le code effraie certains débutants de l'apprentissage de la programmation iOS.

La publication du langage de programmation Swift est probablement la réponse d'Apple à certains de ces commentaires. La syntaxe est beaucoup plus propre et plus facile à lire. Ce langage accélère aussi le processus de développement en augmentant la productivité. Swift attire plus de développeurs web pour créer des applications. Une fois que vous vous habituez à la programmation Swift, il serait vraiment difficile pour vous de revenir à Objective-C.

Si vous venez du développement Web et que vous avez une certaine expérience de programmation sur les langages de script, le passage à Swift sera assez facile. Cela dit, même si vous êtes un débutant total sans expérience de programmation préalable, vous trouverez également le langage assez familier et avec lequel vous allez être rapidement à l'aise.

En Juin 2015, Apple a annoncé Swift 2 et l'ouverture du langage de programmation en open source, ce qui est assez énorme. Depuis, les développeurs ont créé des projets open source intéressants et étonnants en utilisant Swift. Non seulement vous pouvez utiliser Swift pour développer des applications iOS, mais des entreprises comme IBM développé des frameworks web pour créer des applications Web avec Swift. L'engouement pour Swift est sans précédent et même Google, citant le langage comme un langage "de première classe", pourrait utiliser Swift dans son système Android.

En Juin 2016, Apple a introduit Swift 3. La nouvelle version du langage de programmation est intégrée dans Xcode 8, qui est publié en Septembre 2016. C'est l'une des plus grandes sorties depuis la naissance du langage. Certaines API ont été renommées et d'autres fonctionnalités ont été introduites. Tous ces changements aident à rendre le langage encore meilleur, et permettent au développeur d'écrire du code plus beau.

Il n'est pas possible d'apprendre, toute la programmation en Swift en quelques heures. Ruby, PHP, Java ou encore JavaScript sont de bons points de départ pour apprendre Swift. Si vous venez d'un fond de développement web, alors vous ne devriez pas avoir de problème d'apprentissage sur les bases de Swift.

## Playgrounds

En 2014, Apple a aussi introduit les Playgrounds dans le cadre de Xcode 6. Ces "terrains de jeux" sont parfaits pour apprendre le langage de programmation Swift. Avec les Playgrounds, vous pouvez vous concentrer sur le code que vous écrivez au lieu de jouer avec les paramètres du projet, les fichiers et les dossiers.

Nous allons créer un Playground pour commencer à apprendre les bases de Swift. Ouvrez Xcode 8 et choisissez New> Playground ... dans le menu File de Xcode. Les Playgrounds prennent en charge OS X, iOS et tvOS. Définissez la plate-forme sur iOS et nommez le projet: "POECPlayground".

Dites à Xcode où vous voulez enregistrer le Playground et cliquez sur Create. À première vue, un Playground semble être un fichier simple avec une extension .playground, mais c'est en fait un dossier avec une poignée de fichiers. Heureusement, ce n'est pas quelque chose dont nous devons nous inquiéter.

L'interface utilisateur d'un Playground est propre et simple. Vous pouvez voir l'éditeur de code sur la gauche et l'output sur la droite. Xcode a déjà rempli l'éditeur avec trois lignes de code.

//: Playground - noun: a place where people can play

La ligne commence par deux barres obliques, //, indiquant que cette ligne est un commentaire et ignorée par le compilateur.   
Les commentaires qui s'étendent sur plusieurs lignes commencent par /\* et finissent par \*/.

/\*

 This is a longer

 comment spanning

 multiple lines.

 \*/

Xcode a également ajouté une instruction import pour le framework UIKit. Nous verrons plus tard que UIKit est une des librairies les plus importantes pour le développement sur iOS. Dans Swift, importer une librairie est aussi simple que d'utiliser le mot-clé import suivi du nom du framework correspondant.

import UIKit

La dernière ligne peut rappeler du JavaScript. Swift utilise le mot-clé var pour déclarer une variable, str, et lui assigner une valeur, "Hello, playground". Cet exemple montre immédiatement une caractéristique clé de Swift, l'inférence de type.

var str = "Hello, playground"

L'exemple ci-dessus est un peu trompeur. Vous pouvez penser que Swift est un langage faiblement typé, mais il n'en est rien. La variable str est de type String. Swift est assez intelligent pour déduire le type de str en regardant la valeur qui lui est affectée.

L'exemple suivant est équivalent. Dans cet exemple, le type String de la variable str est explicitement spécifié.

var str: String = "Hello, playground"

## Variables et constantes

Swift définit également le mot-clé let. Ce mot clé est utilisé pour déclarer une constante. Le langage de programmation Swift encourage l'utilisation de constantes dans la mesure du possible. Si la valeur d'une variable ne change pas pendant sa durée de vie, elle devrait être une constante. Comme son nom l'indique, une fois qu'une constante est donnée une valeur, elle ne peut pas être modifiée.

Dans le cas d'une variable non modifiée, Xcode vous avertira afin de vous inciter à utiliser une constante. Une erreur sera indiquée lors d'une modification d'une constante.

Les noms de variables et constantes ne doivent pas contenir d'espaces et peuvent commencer une lettre ou un \_. Autre particularité du langage, il est possible d'utiliser les emojis au même titre qu'un caractère.

## Types de données

Swift définit un large éventail de types de données, mais nous allons jeter un oeil à certains des types de données les plus courantes que vous rencontrerez.

### Booléens

Commençons par un type simple, Bool. Les booléens de Swift sont tout aussi simples et faciles à utiliser que dans d'autres langues.

let myBool = true

var notTrue = false

notTrue = true

### Numbers

Les nombres sont représentés par Int, UInt, Float, ou Double[[2]](#footnote-2). Il existe d'autres types de données pour stocker des valeurs numériques, mais celles-ci sont les plus courantes. Ces types de données sont similaires à ceux que vous trouvez dans d'autres langages de programmation.

Parce que Swift est un langage fortement typé, vous avez besoin de traiter les conversions de type de temps en temps. Par exemple, il n'est pas possible de multiplier un Int et un Float.

let a : Int = 2

let b : Float = 2.65

let c = Float(a) \* b

Même si on peut penser caster en Float, ce n'est pas ce qui se passe. Nous créons (initialisons) un Float en utilisant la valeur stockée dans a.

Il existe deux façons de créer une constante ou une variable. Les constantes a et b ont été créées en utilisant la syntaxe littérale. La variable str dont nous avons discuté précédemment est un autre exemple de syntaxe littérale.

Pour créer un Float à partir de la valeur stockée dans a, nous utilisons la syntaxe d'initialisation. Nous créons le Float en utilisant le nom du type, Float, suivi de deux parenthèses, en passant en a comme paramètre.

### Strings

Nous avons déjà rencontré certaines chaînes précédemment. Une chaîne n'est rien de plus qu'une séquence de caractères. En fait, il est possible d'accéder à la collection de caractères comme indiqué dans l'exemple ci-dessous.

let company = "Coaxys"

let companyInit = String("Coaxys")

let chars = company.characters

Dans l'exemple ci-dessus, nous utilisons une chaîne littérale. L'exemple suivant illustre comment utiliser l'initialiseur du type String. La syntaxe est identique à ce que nous avons vu dans un des exemples précédents sur les nombres et la conversion de type.

### Collections

La bibliothèque standard Swift défini trois types de collections majeures : Array, Dictionary et Set. Ces types de collections nécessitent peu d'explications si vous êtes familiers avec un autre langage de programmation, dans ces derniers ils peuvent être nommés Hash, Map ou List.

let myArray = [1, 2, 3, 4, 5]

let myDictionary = ["One": 1, "Two": 2, "Three": 3]

let mySet = Set(["One", "Two", "Three", "Four", "Five"])

var arr1 : [Int] = []

arr1.append(2)

let arr2 : [Any] = [1, 2, 3, "Four", "Five"]

Pour rappel, Swift est un langage fortement typé et cela a également des conséquences pour les types de collection. La variable arr1 est de type [Int], cela signifie que arr1 ne peut contenir que des valeurs de type Int.

### Any et AnyObject

La bibliothèque standard Swift définit deux types spéciaux, Any et AnyObject. Le type Any peut représenter n'importe quel type tandis que le type AnyObject peut représenter n'importe quelle classe ou structure. Cela peut paraitre confus : Swift a été conçu pour être fortement typé et à première vue Any et AnyObject réduisent la sécurité de type.

Any et AnyObject ont principalement été introduits pour des raisons de compatibilité. En effet, la plupart des frameworks utilisés pour le développement sur iOS sont écrits en C et Objective-C. Objective-C définit le type id, qui représente tout objet. Pour s'assurer que Swift et les frameworks Cocoa puissent travailler ensemble, la bibliothèque standard Swift a ainsi introduit les types Any et AnyObject.

## Classes et Structures

### Classes

Si vous êtes familiers avec la programmation orientée objet alors vous devez déjà connaitre les classes. Les classes sont très simples en Swift. L'exemple suivant défini un classe Boat avec deux propriétés : speed et lifeboats ainsi qu'une méthode : deployLifeboats().

class Boat {

    var speed: Float = 0

    var lifeboats: Int = 2

    func deployLifeboats() {

        // ...

    }

}

L'initialisation et la modification d'une instance de la classe Boat décrit ci-dessous, tout comme l'appel d'une méthode de classe.

var boat = Boat()

boat.speed = 10.5

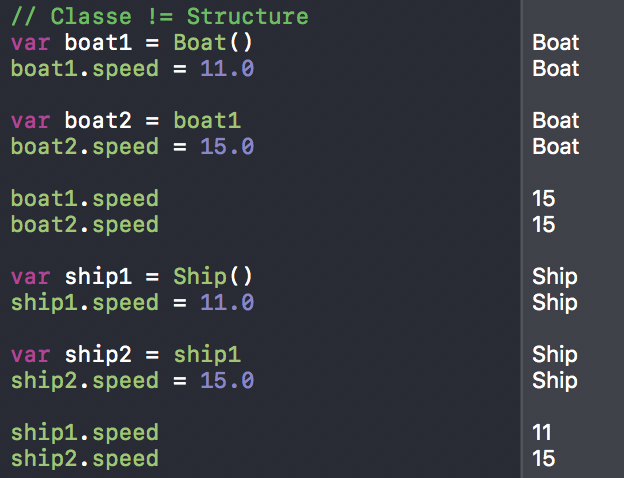
boat.lifeboats += 1

boat.deployLifeboats()

### Structures

Ormis le mot clé struct qui remplace class, une structure se défini de la même façon qu'une classe.

### Classe vs Structure



L'exemple ci-dessus montre que les classes sont passées par référence tandis que les structures sont passées par valeur. Nous avons initialisé une instance de Boat, l'avons affecté à boat1, et avons défini sa propriété de speed à 11.0. Nous avons assigné le boat1 au boat2. Cela signifie que boat2 a une référence à boat1. Le boat1 et le boat2 pointent la même instance de Boat. Le résultat est que le réglage de la propriété speed de boat2 modifie également la propriété speed de boat1.

Ce n'est pas vrai pour les structures. Nous répétons les mêmes étapes en utilisant la structure Ship. Le résultat final est différent cependant. En affectant le ship1 au ship2, la valeur du ship1 est copiée et stockée dans le ship2. Cela signifie que ship1 et ship2 pointent vers différents objets, différentes instances de la structure Ship.

Chaque type primitif vu précédemment est une structure tout comme chaque type de collection.

Il y a d'autres différences qui distinguent les classes et les structures dans Swift. Les classes prennent en charge l'héritage mais pas les structures.

class Speedboat : Boat {

}

var speedboat = Speedboat()

speedboat.speed = 100.0

speedboat.lifeboats = 0

speedboat.deployLifeboats()

Parce que Boat est la classe parente de Speedboat, Speedboat hérite des propriétés et des méthodes définies dans la classe Boat. Même si la mise en œuvre de la classe Speedboat est vide, l'instance de Speedboat possède des propriétés et les mêmes méthodes.

Une différence plus subtile qui distingue classes et structures est la vérification de type à l'exécution. Vous pouvez vérifier le type d'une instance de classe à l'exécution alors que cela n'est pas possible pour les instances de structure.

## Optionals

Les optionnels font partie intégrante du langage de programmation Swift. Tout repose sur le faut qu'une variable doit être initialisée avant de pouvoir être utilisée sinon elle est nulle. Dans de nombreux langages de programmation, les variables ont une valeur initiale par défaut. Par exemple : Objective-C utilisant les pointeurs, certaines variables pouvaient être nulles (NSString \*newString;).

### Qu'est-ce qu'un optional ?

Swift utilise les optionals pour encapsuler un concept important, une variable ou une constante a une valeur ou il n'en a pas. C'est aussi simple chez Swift. Pour déclarer une variable ou une constante comme facultative, nous ajoutons un point d'interrogation au type de la variable ou de la constante.

var optionalStr: String

La variable optionalStr n'est pas du type String. Il est de type optional String. La conséquence est que nous ne pouvons pas interagir directement avec la valeur de optionalStr. La valeur est stockée en toute sécurité dans l'optional et nous devons demander à la valeur facultative qu'elle encapsule.

### Forced unwrapping

Une façon d'accéder à la valeur d'un optionnel est le force unwrap. Nous pouvons accéder à la valeur de optionalStr en ajoutant un "!" au nom de la variable.

Il est important d'être sûr que l'optional contient une valeur avant de faire un unwrap. Si vous forcez un unwrap sur une variable optionnelle qui n'a pas de valeur, Swift propage une erreur et votre application va planter.

### Optional Binding

Il y a une façon plus sécurisée d'accéder à la valeur d'un optional. L'exemple suivant montre comment nous pouvons accéder en toute sécurité à la valeur stockée dans optionalStr, qui est de type String?.

if optionalStr != nil {

    print(optionalStr!)

} else {

    print("optionalStr has no value")

}

Nous vérifions d'abord si optionalStr est égal à nil avant d'afficher son contenu. Dans cet exemple, optionalStr a une valeur. Cela signifie qu'elle ne sera pas unwrapped par accident.

Il existe une approche plus élégante connue sous le nom d'optional binding. Dans l'exemple suivant, nous assignons la valeur stockée dans l'optional à une constante temporaire. La valeur optionnelle optionalStr est liée à la constante strConst et rendue disponible dans la clause if de l'instruction if. Cette approche fonctionne également pour les déclarations while.

if let strConst = optionalStr {

    print(strConst)

} else {

    print("optionalStr has no value")

}

### Qu'est-ce que nil ?

Si vous venez d'un langage comme Objective-C, alors vous savez certainement ce qu'est nil. Dans Objective-C, nil est un pointeur vers un objet qui n'existe pas. Mais Swift définit nil un peu différemment.

Dans Swift, nil signifie l'absence d'une valeur, toute valeur. Alors que nil ne s'applique qu'aux objets dans Objective-C, dans Swift, nil peut être utilisé pour n'importe quel type.

## Où est passé le point-virgule ?

Avez-vous remarqué que nous n'avons pas utilisé de point-virgule dans les exemples précédents ? Dans Swift, il n'est pas nécessaire d'utiliser des points-virgules, car le compilateur est assez intelligent pour comprendre quand une instruction se termine et une autre commence. Si l'utilisation de point-virgule vous manque, alors il est toujours possible et bon de les ajouter. Gardez à l'esprit, cependant, que le guide du langage recommande d'omettre les points-virgules.

## Fonctions

### Qu'est-ce qu'une fonction ?

Une fonction n'est rien de plus qu'un bout de code (une suite d'instructions) qui peut être exécutée quand cela est nécessaire. Voici un exemple comme base de discussion.

func printHelloWorld() {

    print("Hello World!")

}

printHelloWorld()

Une fonction commence par le mot clé func et est suivie du nom de la fonction, printHelloWorld dans l'exemple ci-dessus. Comme dans beaucoup d'autres langages, le nom de la fonction est suivi par une paire de parenthèses qui contiennent les paramètres de la fonction, l'entrée de la fonction.

Le corps de la fonction est enveloppé dans une paire d'accolades. Le corps de la fonction printHelloWorld() contient une instruction dans laquelle nous affichons la chaîne "Hello World!" sur la sortie standard. Pour appeler la fonction, nous tapons son nom suivi d'une paire de parenthèses.

### Paramètres

Nous allons rendre l'exemple ci-dessus un peu plus complexe en ajoutant des paramètres dans la définition de la fonction. En ajoutant des paramètres, nous fournissons à la fonction des valeurs d'entrée qu'elle peut utiliser dans son corps. Dans l'exemple suivant, nous définissons la fonction printMessage(\_ :), qui accepte un paramètre, message, de type String.

func printMessage(message: String) {

    print(message)

}

printMessage(message: "Hello ! 👋")

Une fonction peut accepter plusieurs paramètres ou valeurs d'entrée. Les paramètres sont situés entres les parenthèses qui suivent le nom de la fonction. Le nom du paramètre est suivi par deux points et son type, ceci est identique la déclaration d'une variable ou d'une constante. Il indique simplement que le paramètre message est de type String.

Appeler la fonction est semblable à ce que nous avons vu précédemment. La différence est que nous passons dans un argument.

Il est également possible d'ajouter un \_ devant le nom du paramètre pour utiliser le nommage implicite lors de l'appel de la fonction mais aussi de fixer des valeurs par défaut. Bien évidemment, plusieurs paramètres d'entrée peuvent être renseignés, ils seront séparés par des virgules.

### Types de retour

Les fonctions que nous avons vues jusqu'ici ne nous renvoient rien quand nous les appelons.

Dans l'exemple ci-dessous, nous créons une fonction qui formate une date. La fonction prend deux arguments, une date de type Date (alias de NSDate) et une chaîne de format de type String. Étant donné que les classes Date et NSDateFormatter sont définies dans la librairie Foundation, nous devons importer Foundation en haut du Playground.

func formatDate(date: Date, format: String = "YY/MM/dd") -> String {

    let dateFormatter = DateFormatter()

    dateFormatter.dateFormat = format

    return dateFormatter.string(from: date)

}

let dateAsString = formatDate(date: Date())

Quelques petites explications : le symbole -> indique qua la fonction renvoie une valeur. Le type de la valeur est défini après le symbole ->, ici String.

La fonction prend deux arguments et le second argument a une valeur par défaut. La valeur par défaut de format est "YY/MM/dd". Cela signifie que nous pouvons appeler la fonction avec un seul argument. L'auto-complétion de Xcode prend en charge ce comportement.

Si une fonction n'a pas de type de retour, le symbole -> est omis. C'est pourquoi la fonction printHelloWorld() n'a pas de symbole -> dans sa définition de méthode.

## Closures

### Qu'est-ce qu'une closure ?

Si vous avez travaillé avec des blocs en C et Objective-C ou des closures en JavaScript, vous n'aurez pas de mal à comprendre concept de closures. En réalité, nous avons déjà vu avec des closures précédemment, car les fonctions sont aussi des closures.

Une closure est un bloc d'instructions que vous pouvez transmettre dans votre code. Vous pouvez passer une closure comme un argument d'une fonction ou vous pouvez la stocker comme une propriété d'un objet. Les closures comportent de nombreux cas d'utilisation.

### Syntaxe

La syntaxe basique d'une closure n'est pas difficile.

let closure = {(a: Int) -> Int in

    return a + 1

}

La première chose que vous remarquerez est que la closure entière est enveloppée dans une paire d'accolades. Les paramètres de la closure sont enveloppés dans une paire de parenthèses, séparée du type de retour par le symbole ->. La closure ci-dessus prend un argument, a, de type Int et renvoie un Int. Le corps de la closure commence après le mot-clé in.

Les closures nommées, c'est-à-dire les fonctions globales et imbriquées, sont un peu différentes. Les différences les plus importantes sont l'utilisation du mot clé func et la position des paramètres et du type de retour. Une closure commence et se termine par une accolade, enveloppant les paramètres, le type de retour et le corps. Malgré ces différences, il faut garder à l'esprit que chaque fonction est une closure. Cependant, chaque closure n'est pas une fonction.

### Closures en paramètres

Les closure sont puissantes et l'exemple suivant illustre leur utilité. Ici, nous créons un tableau d'Etats. Nous appelons la fonction map() sur le tableau pour créer un nouveau tableau qui ne contient que les deux premières lettres de chaque Etat sous forme de chaîne majuscule. Une version simplifiée est aussi présente.

var states = ["California", "New York", "Texas", "Alaska"]

let abbreviatedStates = states.map({ (state: String) -> String in

    let index = state.index(state.startIndex, offsetBy: 2)

    return state.substring(to: index).uppercased()

})

let abbreviations = states.map({ state in state.substring(to: state.index(state.startIndex, offsetBy: 2)).uppercased() })

print(abbreviatedStates, abbeviations)

### Trailing closures

Le langage de programmation Swift définit également un concept connu sous le nom de trailing closures. Si vous passez une closure comme dernier argument d'une fonction, vous pouvez placer cette fermeture en dehors des parenthèses de l'appel de fonction. Et si le seul argument de l'appel de fonction est la closure, il est même possible d'omettre les parenthèses de l'appel de la fonction.

L'exemple suivant montre comment cela fonctionne.

let abbreviationsTrailing = states.map { $0.substring(to: $0.index($0.startIndex, offsetBy: 2)).uppercased() }

print(abbreviationsTrailing)

Notez que cela fonctionne également pour les closures qui contiennent plusieurs instructions. En fait, c'est la principale raison pour laquelle les trailing closures sont disponibles dans Swift. Si une closure est longue ou complexe et que c'est le dernier argument d'une fonction, il est souvent préférable d'utiliser la syntaxe de trailing closure.

## Protocols

Les protocoles sont une composante importante du langage de programmation Swift. Le concept n'est pas difficile à comprendre si vous connaissez les protocoles d'Objective-C ou les interfaces en Java. Un protocole définit une conception ou une interface centrée sur une tâche particulière. Pour ce faire, il définit les propriétés et les méthodes nécessaires pour exécuter cette tâche.

La définition d'un protocole ressemble à la définition d'une classe ou d'une structure. Dans l'exemple suivant, nous définissons le protocole Repairable. Le protocole définit deux propriétés, time et cost, et une méthode repair(). La propriété time est en **lecture seule** tandis que la propriété cost est en **lecture / écriture**.

protocol Repairable {

    var time: Float { get }

    var cost: Float { get set }

    func repair()

}

Une classe ou une structure peut adopter un protocole en s'y conformant. Cela signifie qu'il est nécessaire de mettre en œuvre les propriétés et les méthodes définies par le protocole.}

class Boat2: Repairable {

    var speed: Float = 0

    var lifeboats: Int = 2

    var time: Float = 10.0

    var cost: Float = 100.0

    func deployLifeboats() {

        // ...

    }

    func repair() {

        // ...

    }

}

Il est ainsi facile de conformer un type à un protocole. Même si un protocole ne peut pas être instancié comme une classe ou une structure, un protocole est un type valide, il peut ainsi être utilisé comme propriété ou argument de closure.

## Contrôle d'accès

Swift fournit cinq niveaux d'accès différents pour les entités de votre code. Ces niveaux d'accès sont relatifs au fichier source dans lequel une entité est définie, et également par rapport au module auquel ce fichier source appartient.

* Les entités d'accès open et d'accès public peuvent être utilisées dans n'importe quel fichier source à partir de leur module de définition, et également dans un fichier source provenant d'un autre module qui importe le module de définition. Vous utilisez généralement l'accès open ou public lorsque vous spécifiez l'interface public dans un framework. La différence entre l'accès open et public est décrite ci-dessous.
* L'accès internal permet aux entités d'être utilisées dans n'importe quel fichier source à partir de leur module de définition, mais pas dans un fichier source en dehors de ce module. Vous utilisez généralement l'accès internal pour définir la structure interne d'une application ou d'un framework.
* L'accès fileprivate limite l'utilisation d'une entité dans son propre fichier source de définition. Utilisez fileprivate pour masquer les détails d'implémentation d'une fonctionnalité spécifique lorsque ces détails sont utilisés dans un fichier entier.
* L'accès private limite l'utilisation d'une entité à la déclaration jointe. Utilisez l'accès private pour masquer les détails d'implémentation d'une fonctionnalité spécifique lorsque ces détails ne sont utilisés que dans une seule déclaration.

L'accès open est le niveau d'accès le plus élevé (le moins restrictif) et l'accès private est le niveau d'accès le plus bas (le plus restrictif).

L'accès open s'applique uniquement aux classes et aux membres de la classe et diffère de l'accès public comme suit :

* Les classes avec accès public ou tout niveau d'accès plus restrictif ne peuvent être sous-classées que dans le module où elles sont définies.
* Les classes ayant un accès public ou un niveau d'accès plus restrictif peuvent être remplacées par des sous-classes uniquement dans le module où elles sont définies.
* Les classes open peuvent être sous-classées dans le module où elles sont définies et dans n'importe quel module qui importe le module où elles sont définies.
* Les classes open peuvent être remplacées par des sous-classes dans le module où elles sont définies et dans n'importe quel module qui importe le module où elles sont définies.

Le fait de marquer une classe comme open indique explicitement que vous avez considéré l'impact du code provenant d'autres modules en utilisant cette classe comme une superclasse et que vous avez conçu le code de votre classe en conséquence.

# Créer votre première application

Comme énoncé dans les précédentes parties, chaque application iOS passe par un projet Xcode.   
Qu'est-ce qu'un projet Xcode ? Il s'agit d'un conteneur ou un référentiel contenant les fichiers, les ressources et les informations nécessaires à la construction d'un ou de plusieurs produits.

Notez que cela ne se limite pas au code source et aux éléments d'un projet, tels que les images et autres supports. Un projet permet également de suivre les liens entre ses différents éléments et il sait comment construire les produits finaux à partir de ses éléments.

Si cela n'est pas déjà fait, ouvrez Xcode et créez un nouveau projet Xcode en choisissant Nouveau> Projet ... dans le menu Fichier. Vous pouvez accomplir la même chose en appuyant sur Maj + Commande + N.

## Exploration du projet

Il est temps de jeter un œil sur le projet lui-même. Le contenu du projet est affiché dans le navigateur de projet, le premier onglet dans la barre latérale gauche. Sélectionnez le premier élément dans le Navigateur de projet pour afficher les détails du projet dans la vue principale.

La vue principale est composée de deux sections, une barre latérale à gauche et une vue de détail à droite. Dans la barre latérale, vous voyez deux éléments, votre projet avec un élément et des cibles avec un élément.

Il est bon de savoir dès le début quelle est la différence entre un projet et une cible. Rappelez-vous qu'un projet est un référentiel pour les fichiers, les ressources (assets) et les données nécessaires à la construction d'un ou de plusieurs produits. Une cible fait référence à l'un de ces produits. Une cible contient les instructions nécessaires pour construire un produit avec les ressources du projet. Cela signifie qu'un projet peut contenir plusieurs cibles pour construire plusieurs produits. Comme vous pouvez le voir, un projet Xcode est plus qu'un simple dossier avec un tas de fichiers.

Avant de commencer à modifier le code source du projet, il peut être intéressant de compiler et d'exécuter votre nouveau projet pour voir ce que le modèle d'application nous a donné gratuitement. Cliquez sur le bouton Exécuter en haut à gauche et assurez-vous que le schéma actif est configuré pour exécuter l'application dans le simulateur en sélectionnant iPhone 6 ou un autre simulateur pour iPhone.

Si tout allait bien, le simulateur devrait lancer votre application et afficher une vue vide, blanc avec la barre de statut familière en haut.

## Modifier l'interface utilisateur

Vous avez peut-être entendu parler du programme "Hello World" si vous avez déjà lu un livre de programmation. Hello World est un programme pour le premier programmateur à créer. C'est un programme très simple qui produit "Hello, World" sur l'écran d'un périphérique. C'est une tradition dans le monde de la programmation. Nous allons donc commencer par créer une application "Hello World" à l'aide de Xcode.

Malgré sa simplicité, le programme "Hello World" sert à quelques fins :

Il vous donne un aperçu de la syntaxe et de la structure de Swift, le nouveau langage de programmation d'iOS.

Il vous donne également une introduction de base à l'environnement Xcode. Vous apprendrez à créer un projet Xcode et à définir votre interface utilisateur à l'aide de Interface Builder. Même si vous avez déjà utilisé Xcode, vous découvrirez les nouveautés de la dernière version de Xcode.

Vous apprendrez comment compiler un programme, construire l'application et le tester à l'aide du simulateur intégré.

1. http://blakespot.com/ios\_device\_specifications\_grid.html [↑](#footnote-ref-1)
2. Double sur 64 bits et Float sur 32 bits – Double plus précis [↑](#footnote-ref-2)