Sumário

[PRIMEIROS PASSO COM O DESENVOLVIMENTO MOBILE EM SWIFT 2](#_Toc5097778)

[O que é Swift 2](#_Toc5097779)

[COMO CONFIGURARA AMBIENTE PARA DESENVOLVER COM SWIFT 3](#_Toc5097780)

[Passo 1. Extrair o arquivo de imagem do macOS Sierra 4](#_Toc5097781)

[Passo 2. Instale o VirtualBox no seu PC 4](#_Toc5097782)

[Etapa 3. Criar nova máquina virtual 4](#_Toc5097783)

[Passo 4. Edite a nova máquina virtual 7](#_Toc5097784)

[Etapa 5 Adicionar código do VirtualBox ao CMD 10](#_Toc5097785)

[Etapa 6 Jogue a máquina virtual 12](#_Toc5097786)

[Etapa 7 Executar uma instalação limpa do macOS Sierra 14](#_Toc5097787)

[Estruturas de ambiente Swift 25](#_Toc5097788)

[A sintaxe básica: 35](#_Toc5097789)

[Comentarios: 35](#_Toc5097790)

[Ponto e virgula 35](#_Toc5097791)

[Identificadores 35](#_Toc5097792)

[Palavra chave 36](#_Toc5097793)

[espaço Swift 36](#_Toc5097794)

[Tipos de Dados em Swift 37](#_Toc5097795)

[Int 37](#_Toc5097796)

[Uint 37](#_Toc5097797)

[Flutuador 38](#_Toc5097798)

[Valor Booleano 38](#_Toc5097799)

[String 38](#_Toc5097800)

[Char 38](#_Toc5097801)

[tipo opcional 38](#_Toc5097802)

[faixa de valores 39](#_Toc5097803)

[**Alias de tipo** 39](#_Toc5097804)

[Tipo de segurança 39](#_Toc5097805)

[Exemplos 41](#_Toc5097806)

[Variáveis em Swift 41](#_Toc5097807)

[Declaração da variável 42](#_Toc5097808)

[Nomenclatura de variáveis 42](#_Toc5097809)

[Variável de saída 42](#_Toc5097810)

[Variável de Tipos Opcionais 43](#_Toc5097811)

[**Forçado a resolver** 44](#_Toc5097812)

[Bind opcional 44](#_Toc5097813)

[Constantes 46](#_Toc5097814)

[Declaração constante 46](#_Toc5097815)

[Operadores Swift 47](#_Toc5097816)

[Operadores aritméticos 47](#_Toc5097817)

[Exemplos 48](#_Toc5097818)

[Comparação 49](#_Toc5097819)

[Exemplos 49](#_Toc5097820)

[Operadores lógicos 50](#_Toc5097821)

[Operadores de atribuição 51](#_Toc5097822)

[Operadores de intervalo 53](#_Toc5097823)

[Outros operadores 55](#_Toc5097824)

[Declaração condicional 56](#_Toc5097825)

[**gramática** 56](#_Toc5097826)

[**fluxograma** 57](#_Toc5097827)

[**Exemplos** 57](#_Toc5097828)

[Swift if ... else 58](#_Toc5097829)

[**gramática** 58](#_Toc5097830)

[**fluxograma** 58](#_Toc5097831)

[**Exemplos** 59](#_Toc5097832)

[Swift if ... else if ... else 59](#_Toc5097833)

[Exemplos 60](#_Toc5097834)

[If aninhados ou encadeados 61](#_Toc5097835)

[**gramática** 61](#_Toc5097836)

[Exemplos 61](#_Toc5097837)

[Switch Case 62](#_Toc5097838)

[**gramática** 62](#_Toc5097839)

[exemplo 1 63](#_Toc5097840)

[Exemplo 2 64](#_Toc5097841)

[Estrutura de Repetição (loops) 64](#_Toc5097842)

[Tipo de loop 65](#_Toc5097843)

[Instruções de controle de loop 65](#_Toc5097844)

[For-In 66](#_Toc5097845)

[Exemplos 66](#_Toc5097846)

[FOR LOOP 67](#_Toc5097847)

[gramática 67](#_Toc5097848)

[Exemplos 68](#_Toc5097849)

[While Loop 69](#_Toc5097850)

[gramática 69](#_Toc5097851)

[Exemplos 70](#_Toc5097852)

[Repeat-While 70](#_Toc5097853)

[gramática 70](#_Toc5097854)

[Exemplos 71](#_Toc5097855)

[Instrução Continue 72](#_Toc5097856)

[gramática 72](#_Toc5097857)

[Exemplos 73](#_Toc5097858)

[Break 73](#_Toc5097859)

[gramática 73](#_Toc5097860)

[Exemplos 74](#_Toc5097861)

[Funções 75](#_Toc5097862)

[**definição de função** 75](#_Toc5097863)

[gramática 75](#_Toc5097864)

[Exemplos 76](#_Toc5097865)

[**chamada de função** 76](#_Toc5097866)

[Argumentos da função 76](#_Toc5097867)

[Função sem parâmetros 77](#_Toc5097868)

[sintaxe: 77](#_Toc5097869)

[Exemplos 77](#_Toc5097870)

[**Tupla para retornar o valor que a função** 77](#_Toc5097871)

[**Nenhum valor de retorno da função** 78](#_Toc5097872)

[**Função Nome do parâmetro** 79](#_Toc5097873)

[Nome do parâmetro local 79](#_Toc5097874)

[nome do parâmetro externo 79](#_Toc5097875)

[funções aninhadas 80](#_Toc5097876)

[Iniciando o estudo de OO em Swift 81](#_Toc5097877)

[Classe 81](#_Toc5097878)

[**tipo de corpo e comparação da estrutura** 81](#_Toc5097879)

[sintaxe: 81](#_Toc5097880)

[definição de classe 82](#_Toc5097881)

[Exemplos 82](#_Toc5097882)

[Métodos 83](#_Toc5097883)

[Métodos da instância 83](#_Toc5097884)

[gramática 83](#_Toc5097885)

[Exemplos 83](#_Toc5097886)

[Construtores / Inicializadores 84](#_Toc5097887)

[**A atribuição inicial da propriedade de tipo de armazenamento** 85](#_Toc5097888)

[**construtor** 85](#_Toc5097889)

[gramática 85](#_Toc5097890)

[Exemplos 85](#_Toc5097891)

[**Parâmetros de configuração** 86](#_Toc5097892)

[Processo de herança e construção da classe 87](#_Toc5097893)

[Especifica o construtor de instância 88](#_Toc5097894)

[Herança Swift 89](#_Toc5097895)

[classe base 89](#_Toc5097896)

[Subclasse 90](#_Toc5097897)

[Exemplos 90](#_Toc5097898)

[Rewrite (Overriding) 91](#_Toc5097899)

[Substituindo métodos e propriedades 91](#_Toc5097900)

[Substituindo métodos 91](#_Toc5097901)

# PRIMEIROS PASSO COM O DESENVOLVIMENTO MOBILE EM SWIFT

Aqui vamos começar a aprender a programar para o iPhone, aplicativos iOS, para plataforma da Apple, utilizando a linguagem Swift.

No caminho diversas boas práticas e Design Pattern de programação em geral, além de específicos da linguagem Swift.

Teremos a oportunidade de aprender a linguagem e entender o que acontece por trás do que estamos programando. Inicialmente aprenderemos a usar o Story Board e a utilizar a linguagem, conheceremos algumas características positivas e outras que devemos tomar cuidado. Além disso, veremos o básico de introdução a Orientação a Objetos com Swift e poder usar tudo isso em nossas aplicações.

* Pequena introdução sobre a linguagem:

https://www.devmedia.com.br/desenvolvimento-ios-conheca-a-linguagem-swift/31860

# O que é Swift

A **linguagem da Apple chamada Swift** está cada vez mais sendo utilizada. Muitos já estão criando ou modificando seus projetos e aprendendo as vantagens que a linguagem traz. Este artigo mostra na teoria e na prática as melhorias que Swift apresenta para os desenvolvedores, principalmente para aqueles que estão iniciando seus projetos iOS e qual a melhor forma de adquirir conhecimento na linguagem.

Há poucos meses a Apple divulgou na [**WWDC**](https://developer.apple.com/wwdc) (WorldWide Developers Conference) a sua **nova linguagem chamada Swift** deixando muitas pessoas surpresas e até aquele momento não ocorreu nenhum vazamento de informações por nenhum meio de comunicação, mesmo assim a linguagem estava há anos sendo desenvolvida em segredo.

Comparada com **[Objective-C](https://www.devmedia.com.br/introducao-ao-objective-c/23061" \t "_blank" \o "Introdução ao Objective C)** que está há mais de 20 anos no mercado, **Swift traz facilidades e característica que a antiga linguagem da Apple não tem** como **[Generics](https://www.devmedia.com.br/orientacao-a-objetos-com-ruby/33726" \t "_blank" \o "Usando Generics em Java)** e sintaxes mais simples onde com poucas linhas de código o programador consegue fazer mais coisas do que faria com Objective-C, lembrando linguagens de script como [**Ruby**](https://www.devmedia.com.br/usando-generics-em-java/28981) e [**Python**](https://www.devmedia.com.br/curso/curso-python/426), porém mantendo as chamadas dos métodos principais parecidas com Objective-C facilitando assim a migração de novos e antigos desenvolvedores.

O Rivest Cipher 4 é uma cifra de encriptação inventada em 1987 por Ron Rivest, um dos inventores do algoritmo de chave pública [**RSA**](https://www.devmedia.com.br/forum/desenvolvendo-criptografia-rsa-em-java/562444) e co-fundador da segurança RSA. Essa cifra de encriptação teve muito sucesso e graças a sua simplicidade e eficiência foi usada em vários protocolos populares como [**WEP, WPA**](https://www.devmedia.com.br/seguranca-em-redes-ieee-802-11-revista-infra-magazine-7/25680), [**SSL**](https://www.devmedia.com.br/forum/utilizacao-de-ssl/411235) ou TLS. Infelizmente, ela tem algumas fraquezas e não é mais usada nos protocolos modernos. O algoritmo RC4 gera uma chave pseudo-random que é usada para gerar o texto cifrado. Esta chave é chamada de pseudo-random porque gera uma sequência de números que se aproximam de propriedades randômicas, ou seja, parece aleatório mas não é já que todo output é o mesmo caso seja passado o mesmo input. A Apple implementou o RC4 nas linguagens Objective-C, Python e Swift e comparou a velocidade de execução de cada uma delas. **O estudo comparativo mostrou que a Swift é 220 vezes mais rápida que Python** e 93 vezes mais rápida que Objective-C.

Mas como já sabemos, temos algumas limitações de compatibilidade de hardware e software no desenrolar deste aprendizado. Isso porque a linguagem Swift faz parte do universo Apple, assim como todos os dispositivos que podemos criar nossas aplicações. Então para que seja possível utilizar a linguagem Swift e criar soluções para IOS vamos precisar fazer alguns ajustes e configurações. Segue abaixo um mini manual de como configurar seu ambiente de desenvolvimento.

Para criar aplicativos no Mac, é necessário um SDK chamado Xcode. O Xcode é um ambiente de desenvolvimento integrado (IDE) que consiste em um conjunto de ferramentas de desenvolvimento de software projetadas pela Apple especificamente para desenvolvimento de software no Mac OS X e no iOS. O kit de desenvolvimento de aplicativos, o Xcode, não está disponível para nenhum outro sistema operacional, exceto Mac OS X e iOS. Há muitas razões e problemas de compatibilidade por trás dessa indisponibilidade.

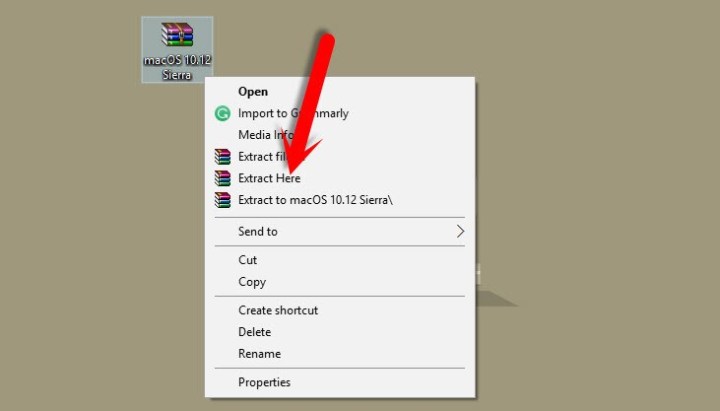
**PS: Para que prestou atenção nas aulas de virtualização do professor Wilhelm Klaus semestre passado terá mais facilidade. Rs.**

COMO CONFIGURARA AMBIENTE PARA DESENVOLVER COM SWIFT:

* Será necessário primeiramente fazer a instalação do programa Oracle Virtual Box ou VMware para que possamos virtualizar um ambiente MAC. Segue o link: *https://www.virtualbox.org/wiki/Downloads* ou acesse a pasta compartilhada (Nuvem-Etec > Luis > Mobile) e copie o .exe
* Após fazer a instalação do virtualizador será necessário fazer download da ISO do MAC-OS. Para que seja compatível rodar a última versão do Xcode e do Swift necessitaremos de utilizar uma ISO na versão 10.13.6 (high Sierra) ou superior. A ISO está disponível também na pasta compartilhada ou pode ser feito o download no link: [CLIQUE AQUI](https://doc-04-b4-docs.googleusercontent.com/docs/securesc/dnqr8ge7fpjr9fkmt135rtod9um6c2ds/l2sr2oekj7vom4603sd8g1rmo460tqr4/1546048800000/12005652994417233588/14294032850204428226/1K3MagEB3-hly1AdDCVBHYynsAFIw-qLV?e=download&nonce=e24o8pu8qvidc&user=14294032850204428226&hash=e0m81joom9qntctn0od8qe8jdepjv33p).
* Após fazer o download crie a máquina virtual seguindo os seguintes passos:

## **Passo 1. Extrair o arquivo de imagem do macOS Sierra**

Depois de baixar o arquivo de imagem do macOS Sierra, você deve extraí-lo via [WinRAR](http://www.win-rar.com/download.html?&L=0) ou [7zip](http://www.7-zip.org/download.html) . Para extrair o arquivo, basta clicar com o botão direito do mouse no arquivo e escolher " **Extrair aqui** ". Isso levará alguns minutos para ser extraído. Aguarde um momento.

[](https://i0.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Extact-macOS-Sierra-VM-File.jpg)

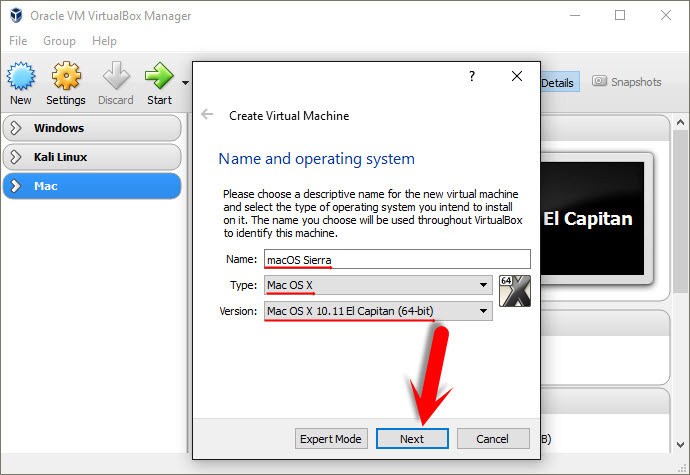
Extraia o macOS Sierra VM File

## **Passo 2. Instale o VirtualBox no seu PC**

Baixe o VirtualBox no link acima. Depois de ter baixado o VirtualBox, instale-o no seu PC Windows. Para instalar o VirtualBox, basta clicar sobre a configuração, em seguida, clique em " **Next** " e " **Next** ". Finalmente, clique em " **Concluído** ".

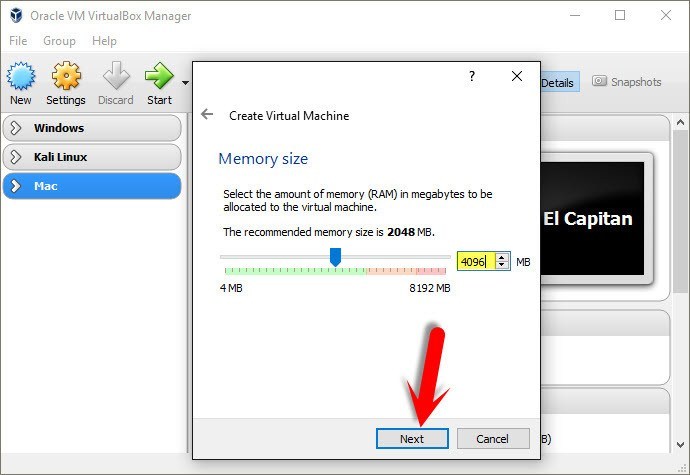
## **Etapa 3. Criar nova máquina virtual**

**# 1** . Agora, você precisa criar uma nova máquina virtual. Abra o VirtualBox, em seguida, toque em " **Novo** " no canto superior esquerdo e nomeie a máquina virtual " **macOS Sierra** ", em seguida, pressione o botão " **Next** ".

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Name-and-choose-the-Version.jpg)

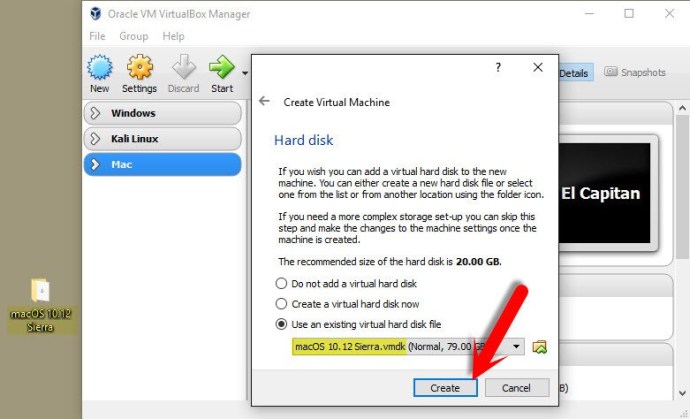
Nomeie e escolha a versão

**# 2** . Agora, você será solicitado a selecionar o tamanho da memória para a VM. Escolha pelo menos **4 GB de** espaço **de RAM** para a máquina virtual.

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Choose-Memory-Size.jpg)

Escolha o tamanho da memória

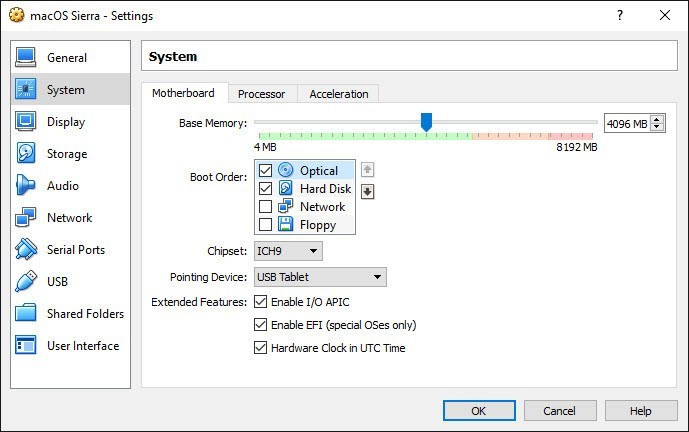
**# 3** . Se desejar, você pode adicionar um disco rígido virtual à nova máquina. Você pode criar um novo arquivo do disco rígido ou selecionar um do último ou outro local usando o ícone da pasta. Aqui, você deve escolher a última opção " **Usar um arquivo de disco rígido virtual existente** ". Selecione o arquivo e clique em " **Criar** ".

[](https://i2.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Select-Hard-Disk-1.jpg)

Selecione o tipo de disco rígido

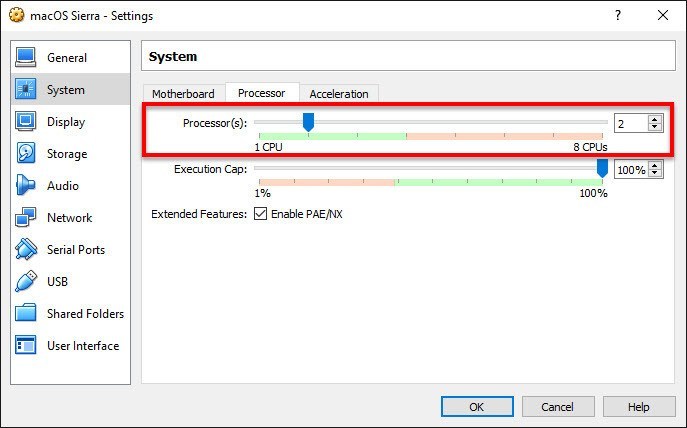
## **Passo 4. Edite a nova máquina virtual**

**# 1** . Abra o VirtualBox e toque em " **Configurações** ". Na " **Guia Geral** ", coloque tudo no padrão.   
Vá para a próxima opção " **Sistema** ". Você verá três guias principais. Clique na primeira aba “ **Motherboard** “. Desmarque a unidade **“Floppy”** e defina o chipset para “ **ICH9** “.

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Motherboard-Settings.jpg)

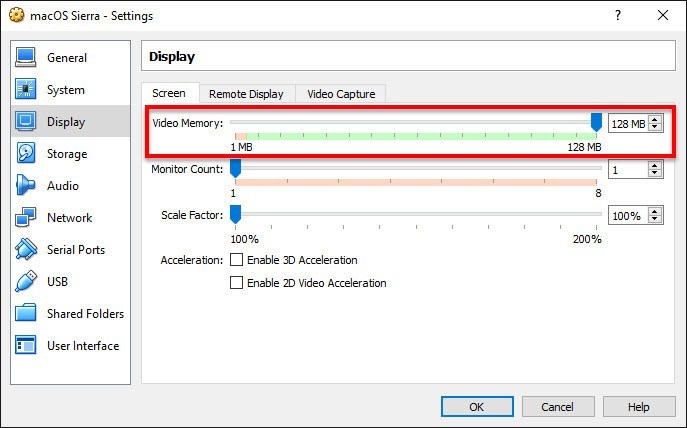
Configurações da placa-mãe

**# 2.**Escolha dois processadores principais ou mais de dois.

[](https://i2.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Choose-Processor-amount.jpg)

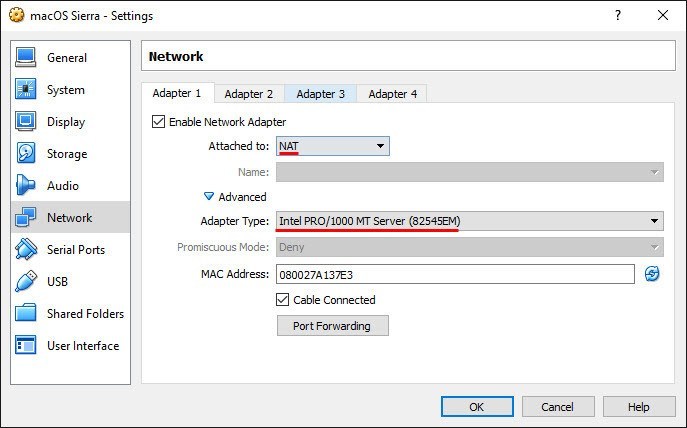
Selecione o valor do processador

**# 3** . Navegue até “ **Opção de exibição** ” e coloque a memória de vídeo em **128 MB** .

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Choose-Video-Memory.jpg)

Escolha a memória de vídeo

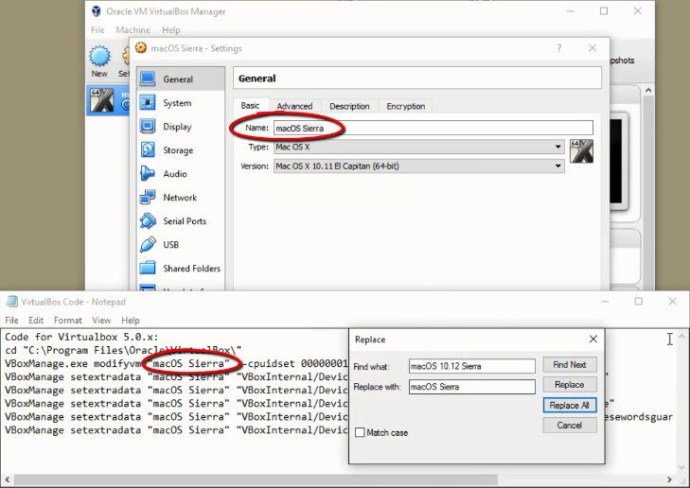
**# 4.**Agora, navegue até a guia **"Rede" e** depois **"Ativar adaptador de rede"** . E escolha o tipo de adaptador **“Intel PRO / 1000 MT Server (82545EM)”** .

[](https://i0.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Enable-Network-Adapter.jpg)

Ativar adaptador de rede

## **Etapa 5 Adicionar código do VirtualBox ao CMD**

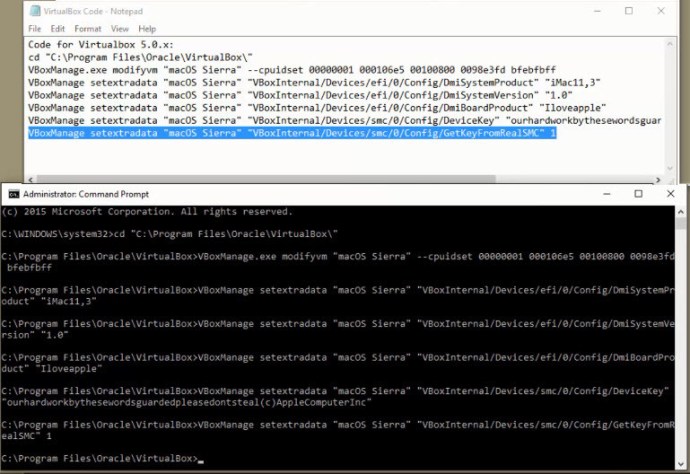
**# 1** . Antes de adicionar o código, você precisa primeiro editar o código. Abra o código do VirtualBox e substitua o **“Virtual Machine Name”** . Obtenha ajuda da captura de tela abaixo.

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Replace-the-Code.jpg)

Substitua o código

**Nota** : Para instalar o macOS Sierra no VirtualBox, você deve “**QUIT** ” VB, caso contrário não funcionará.

**# 2** . Agora, execute o **CMD** como administrador. Abra o código VBox e adicione o código ao CMD, um por um.

[](https://i0.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Copy-and-Paste-the-VirtualBox-Code.jpg)

*cd "C:\Program Files\Oracle\VirtualBox\"*

*VBoxManage.exe modifyvm "MacOS 10.13" --cpuidset 00000001 000106e5 00100800 0098e3fd bfebfbff*

*VBoxManage setextradata "MacOS 10.13"*

*"VBoxInternal/Devices/efi/0/Config/DmiSystemProduct" "iMac11,3"*

*VBoxManage setextradata "MacOS 10.13" "VBoxInternal/Devices/efi/0/Config/DmiSystemVersion" "1.0"*

*VBoxManage setextradata "MacOS 10.13" "VBoxInternal/Devices/efi/0/Config/DmiBoardProduct" "Iloveapple"*

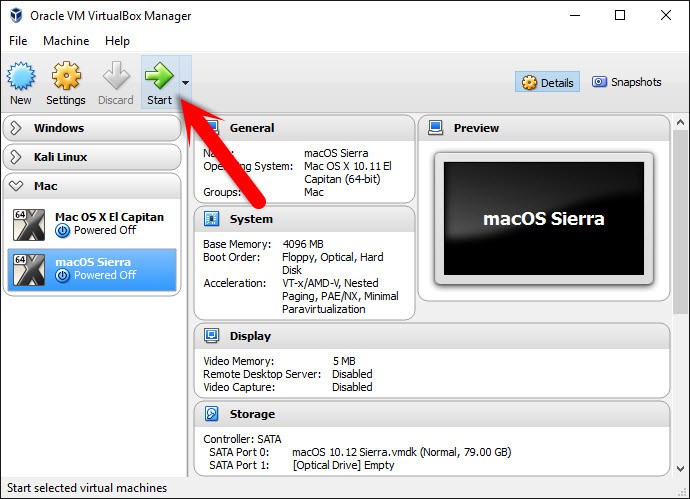
*VBoxManage setextradata "MacOS 10.13" "VBoxInternal/Devices/smc/0/Config/DeviceKey" "ourhardworkbythesewordsguardedpleasedontsteal(c)AppleComputerInc"*

*VBoxManage setextradata "MacOS 10.13" "VBoxInternal/Devices/smc/0/Config/GetKeyFromRealSMC" 1*

Copie e cole o código do VirtualBox

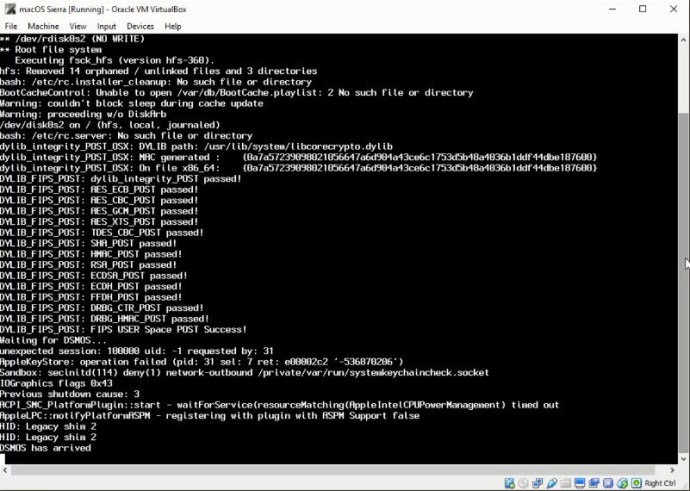
## **Etapa 6 Jogue a máquina virtual**

**# 1** . Nesta etapa, você deve abrir o VirtualBox e depois **“Iniciar”** ou executar o macOS Sierra VM.

[](https://i0.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Start-the-Virtual-Machine.jpg)

Lançar a máquina virtual

**# 2** . Depois de executar o macOS Sierra VM, você verá alguns códigos em branco em uma tela preta. Você não vai fazer nada nesta etapa. Apenas espere um momento e a página de boas-vindas macOS Sierra aparecerá.

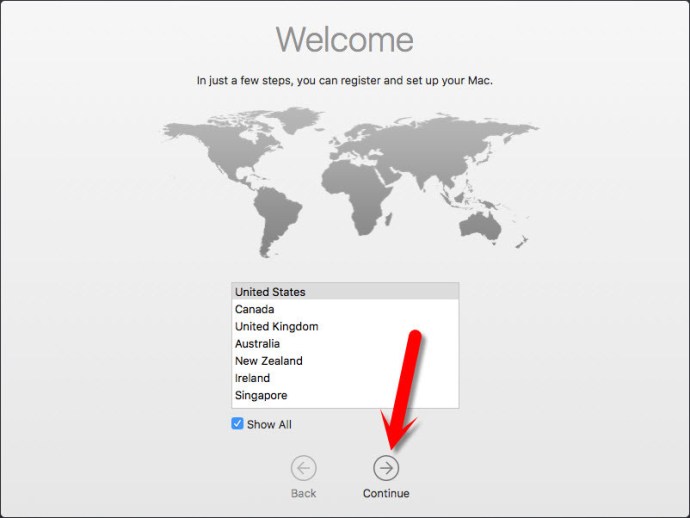
[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Installing-macOS-Sierra-on-VirtualBox.jpg)

Instalando o macOS Sierra no VirtualBox

## **Etapa 7 Executar uma instalação limpa do macOS Sierra**

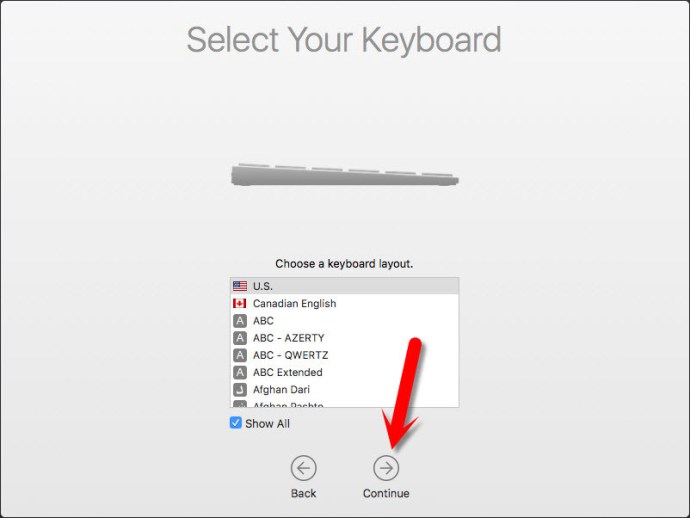
Agora, enquanto instalamos o macOS Sierra ou El Capitan no VMware ou VirtualBox, não precisamos instalá-lo em disco separado, pois ao editar a VM, escolhemos o tipo de disco virtual SATA, e essa é a única maneira de instalar o macOS agora Sierra ou El Capitan no VMware ou VirtualBox. Então é por isso que agora, na próxima etapa, temos que realizar uma instalação limpa e configurar o macOS Sierra.

**# 1.**Escolha o seu **país** e clique em **Continuar** .

[](https://i2.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Choose-Language.jpg)

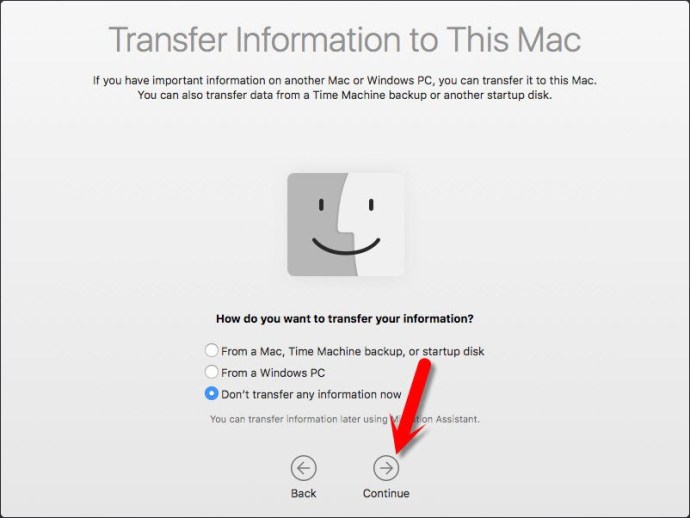
Escolha o seu idioma

**# 2.**Escolha um Layout de **teclado** e clique em **Continuar** .

[](https://i0.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Choose-Keyboard-Layout.jpg)

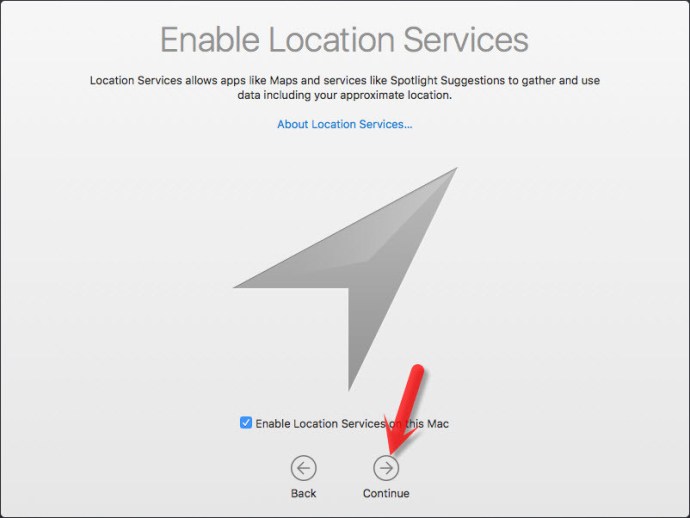
Escolha o layout do teclado

**# 3.**Agora selecione seu método para **transferir informações** para este Mac e clique em **Continuar** .

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Transfer-Information-to-this-Mac.jpg)

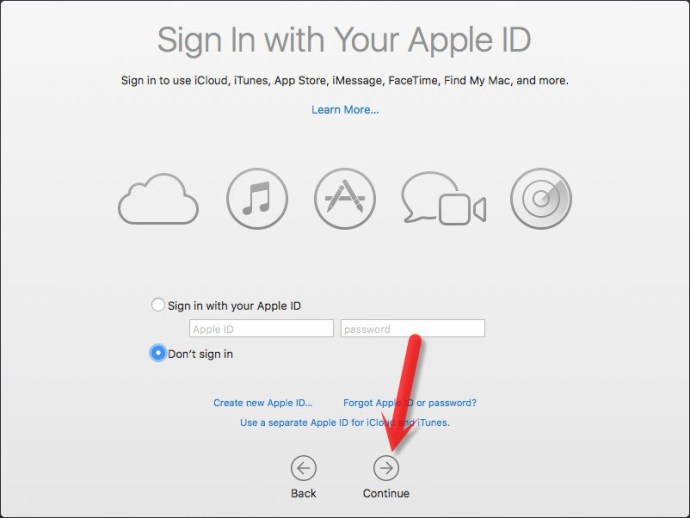
Transferir informações para este Mac

**# 4. Ative os serviços de localização** e clique em **Continuar** .

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Enable-Location-Services.jpg)

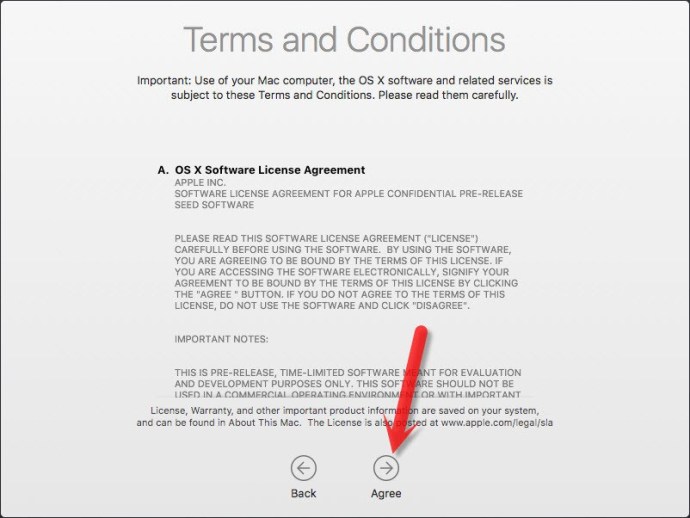
Ativar serviços de localização

**# 5.**Nesta etapa, você precisa fazer **login com sua ID da Apple** . Se você tiver um, crie um ID Apple grátis, pelo artigo,  [como criar um ID Apple no iPhone, iPad ou iPod Touch.](https://www.wikigain.com/create-apple-id-without-credit-card-using-paypal/) Se você não quiser fazer login com a ID da Apple ou não quiser criar a ID da Apple, escolha **Não fazer login** e clique em **Continuar** .

[](https://i2.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Sign-in-with-your-Apple-ID.jpg)

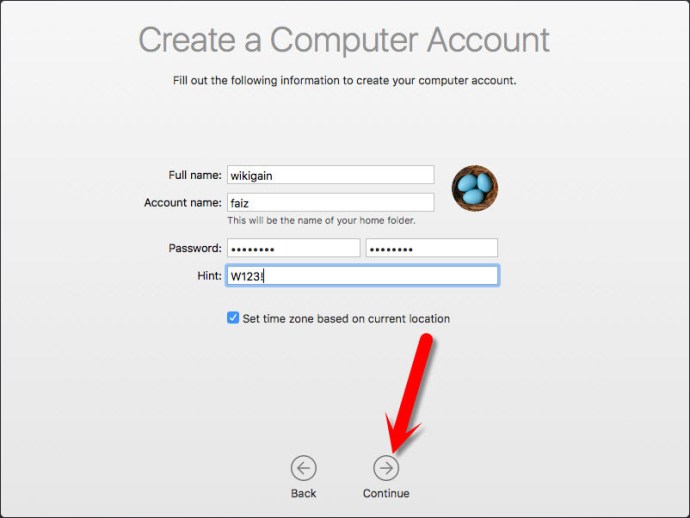
Entre com seu ID da Apple

**# 6. Concorde** com os **Termos e Condições** .

[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Terms-and-Conditions.jpg)

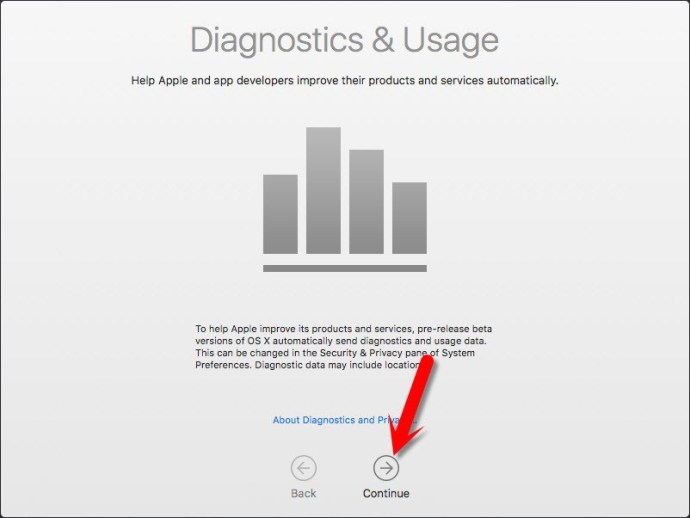
Termos e Condições

**# 7.**Preencha o **nome completo, nome da conta, senha, dica** e marque a caixa de **Definir fuso horário com base na localização atual** . Então, finalmente, clique em  **Continuar** para criar sua conta de usuário para o macOS Sierra, que será instalado.

[](https://i2.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Create-a-Computer-Account.jpg)

Crie uma conta de computador

**# 8.**Selecione as opções para **Diagnóstico e Uso** e clique em **Continuar** .

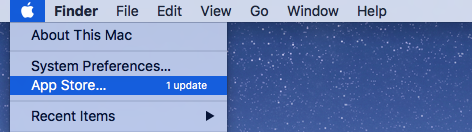
[](https://i1.wp.com/www.Wikigain.com/wp-content/uploads/2016/06/Diagnostics-and-Usage.jpg)

Diagnóstico e Uso

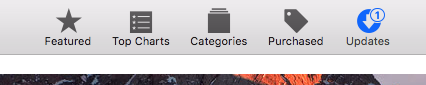
**# 9.**Espere um pouco para **configurar o seu Mac** . Em seguida, ele será executado na área de trabalho do macOS Sierra.

* Após conclusão dos passos acima provavelmente que sua máquina virtual já esteja funcionando o MAC-OS. Então é hora de acessar a AppStore e fazer o download do Xcode. Na AppStore esta disponível apenas a versão mais recente, que só funciona no MAC-OS 10.13.6 ou superior. Caso seu MAC-OS seja inferior, você pode acessar o seguinte link: <https://itunes.apple.com/br/app/macos-mojave/id1398502828?mt=12> e fazer o download de uma versão anterior.

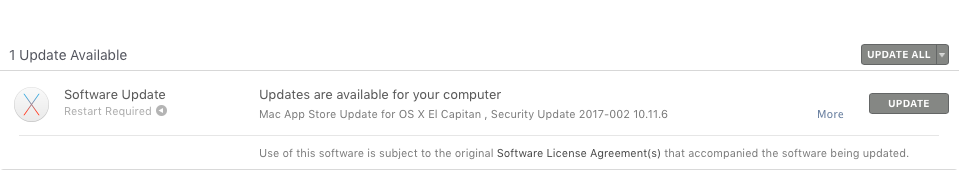
Vamos utilizar a IDE Xcode para desenvolver nossas aplicações com a linguagem Swift, sendo necessário sua instalação para a prática dos exercícios. Mas antes de instalar a IDE própriamente dita é importante atualizar o sistema para evitar possíveis problemas na hora da instalação. Para isso, abra a App Store selecionando a opção correspondente.



Selecione a aba Updates.



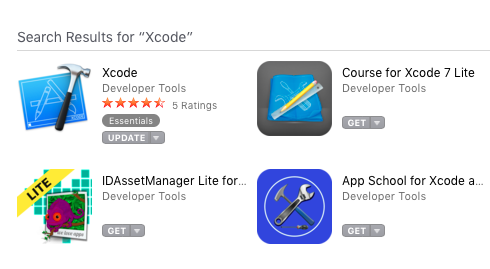
Então verifique se há alguma atualização de sistema pendente, se houver clique no botão Update all à direita da tela. Caso seu sistema já estiver atualizado, basta seguir para a instalação.



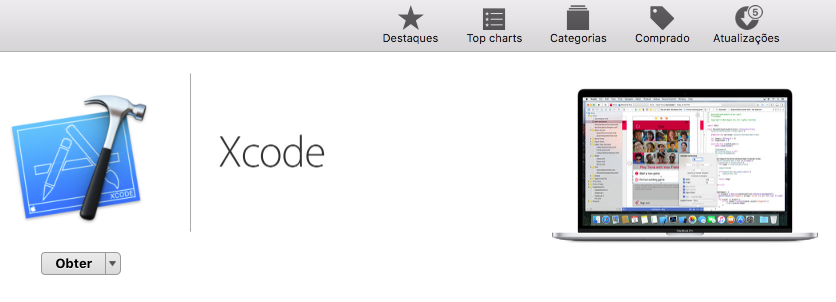
Ainda na App Store, pesquise pelo Xcode no campo de pesquisa, localizado no canto superior direito da página.



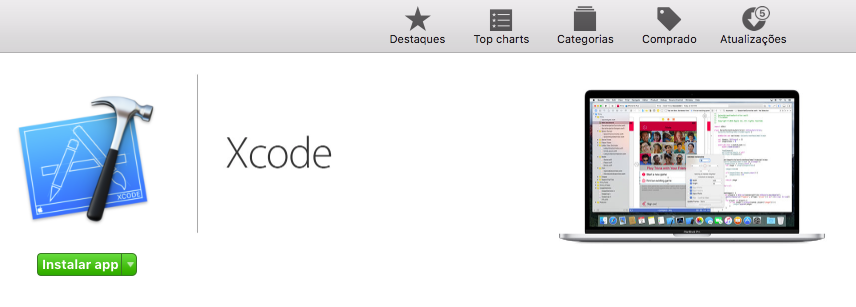
A IDE deve ser a primeira na lista, selecione-o.



Na página do Xcode, clique no botão Obter, abaixo da logo do software.



Após clicar no botão Obter surgirá outro botão, na cor verde, chamado "Instalar app", clique nesse botão e siga as instruções de instalação.



Pronto! Já podemos começar a desenvolver.

# Estruturas de ambiente Swift

Depois que o download estiver concluído, clique duas vezes no arquivo DMG baixado é instalado, a instalação está completa, vamos Xcode ícone para o seu movimento pontapé pasta Aplicativos.



Você também pode procurar o Xcode instalado na App Store, ou no finder do SO.

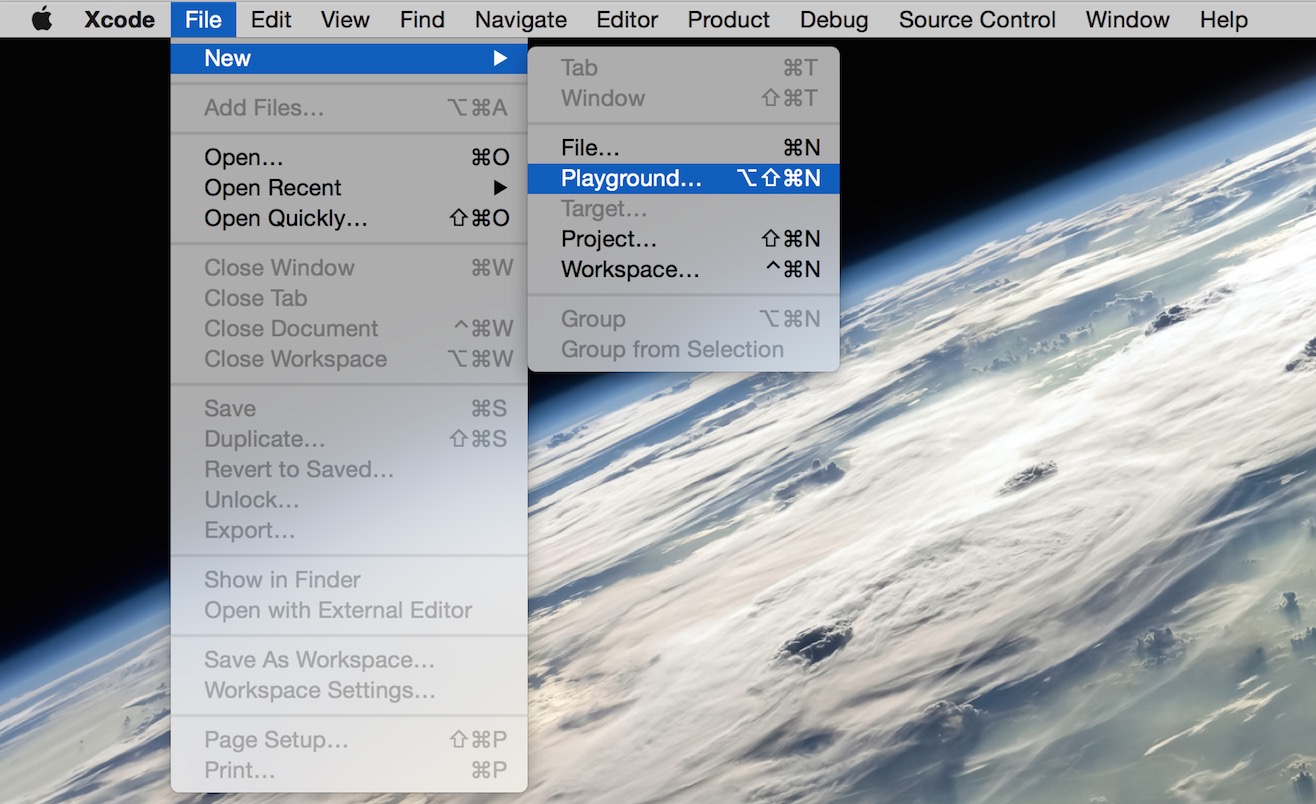
O primeiro programa Swift

Após a instalação estiver concluída, podemos começar a escrever código Swift.

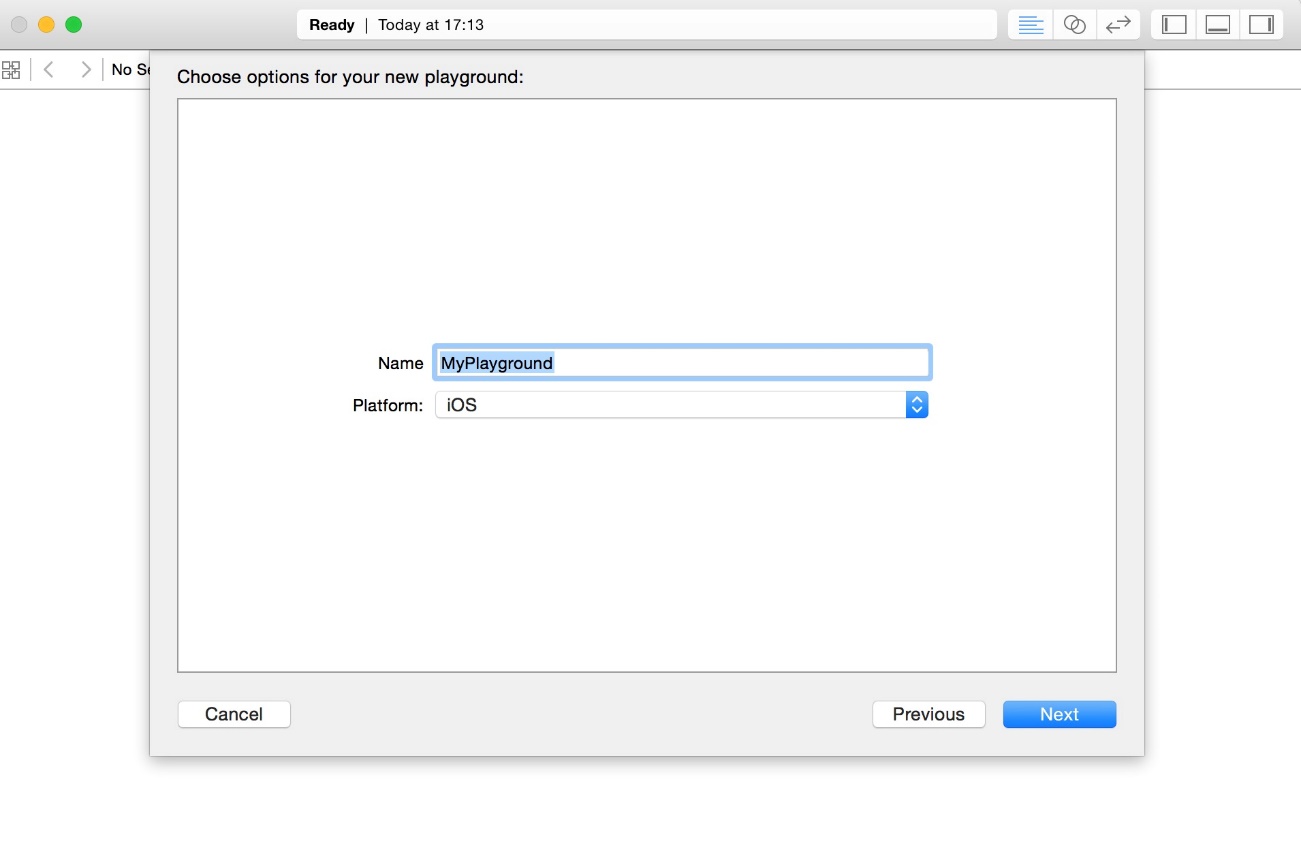
Muito bem, agora que aparentemente está tudo pronto, nosso ambiente, podemos dar início em conhecer a linguagem, a IDE e alguns pontos sobre Swift.

Para facilitar o aprendizado e o desenvolvimento de nossa aplicação, vamos criar um Playground, onde podemos testar nosso código antes de implementá-lo na aplicação.

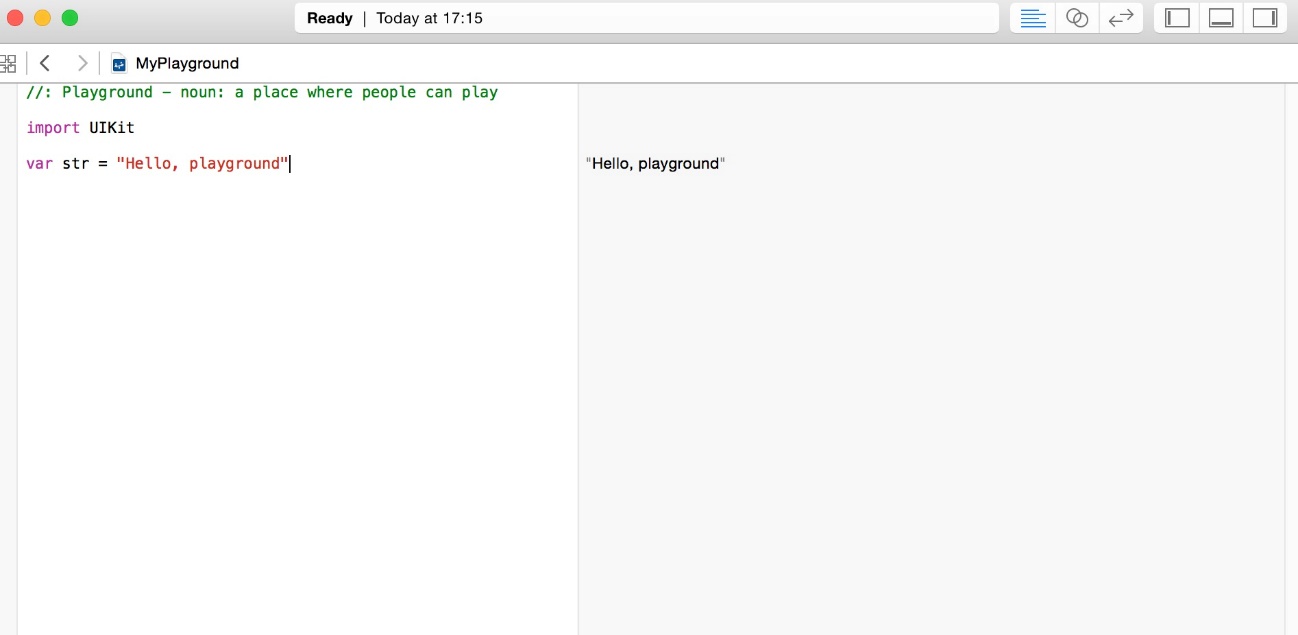
Seguinte, abra a pasta no Xcode arquivo do aplicativo, aberto na parte superior da tela, selecione File => New => Playground.



Em seguida, defina um nome para o pasta de seus projetos e selecionar a plataforma iOS.



O Playground é como um documento interativo, que é usado para a praticar e aprender rápido, e escrever uma linha de código é o resultado (à direita), você pode ver os resultados em tempo real do código, é para aprender de forma rápida.



A seguinte é a janela de código padrão Swift Playground:

import UIKit

var str = "Hello, playground"

Após o programa acima é carregado, ele irá mostrar os resultados do programa na janela Playground direita:

Hello, playground

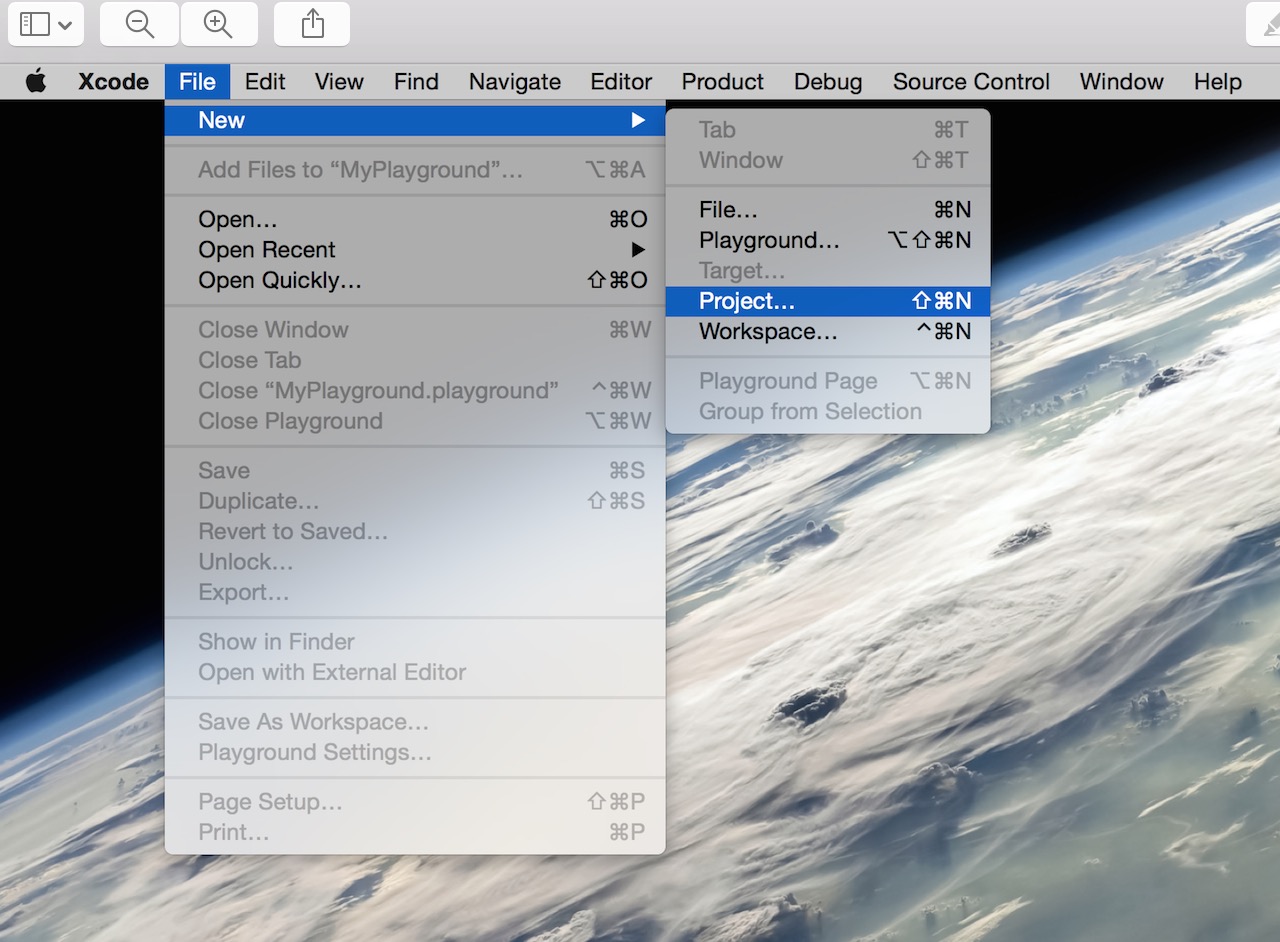
Neste ponto, você já completou o primeiro programa de estudo Parabéns Swift começando.

Criando seu primeiro projeto

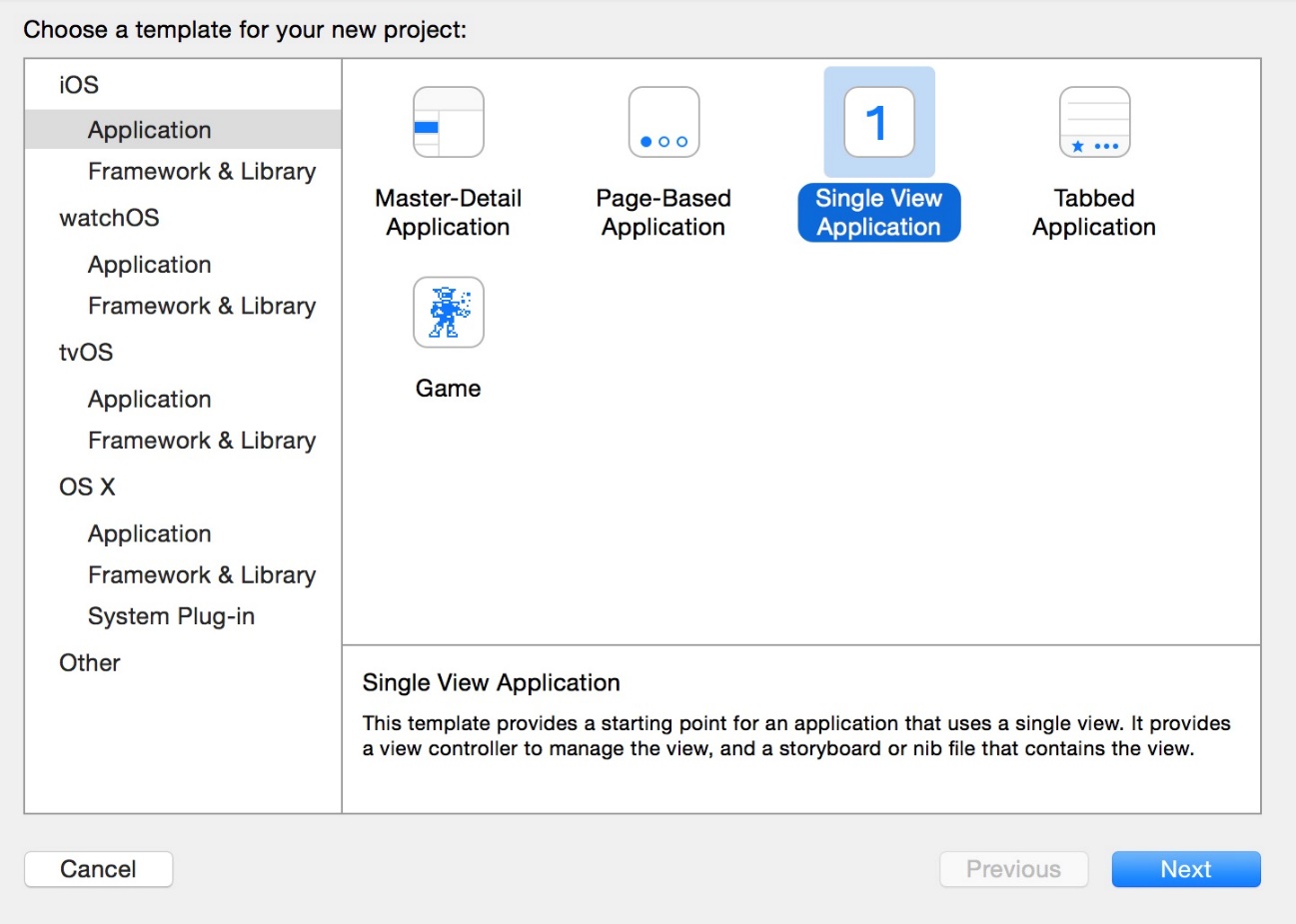
Como dito, o Playground é apenas um local que vamos iniciar nossas primeiras práticas no Xcode. Mas após termos um maior entendimento do funcionamento da linguagem e da ferramenta partiremos para nossos projetos de fato.

Fica já abaixo dicas e passo a passo para quando fomos criar um projeto.

1, abra a ferramenta Xcode, selecione File => New => Project

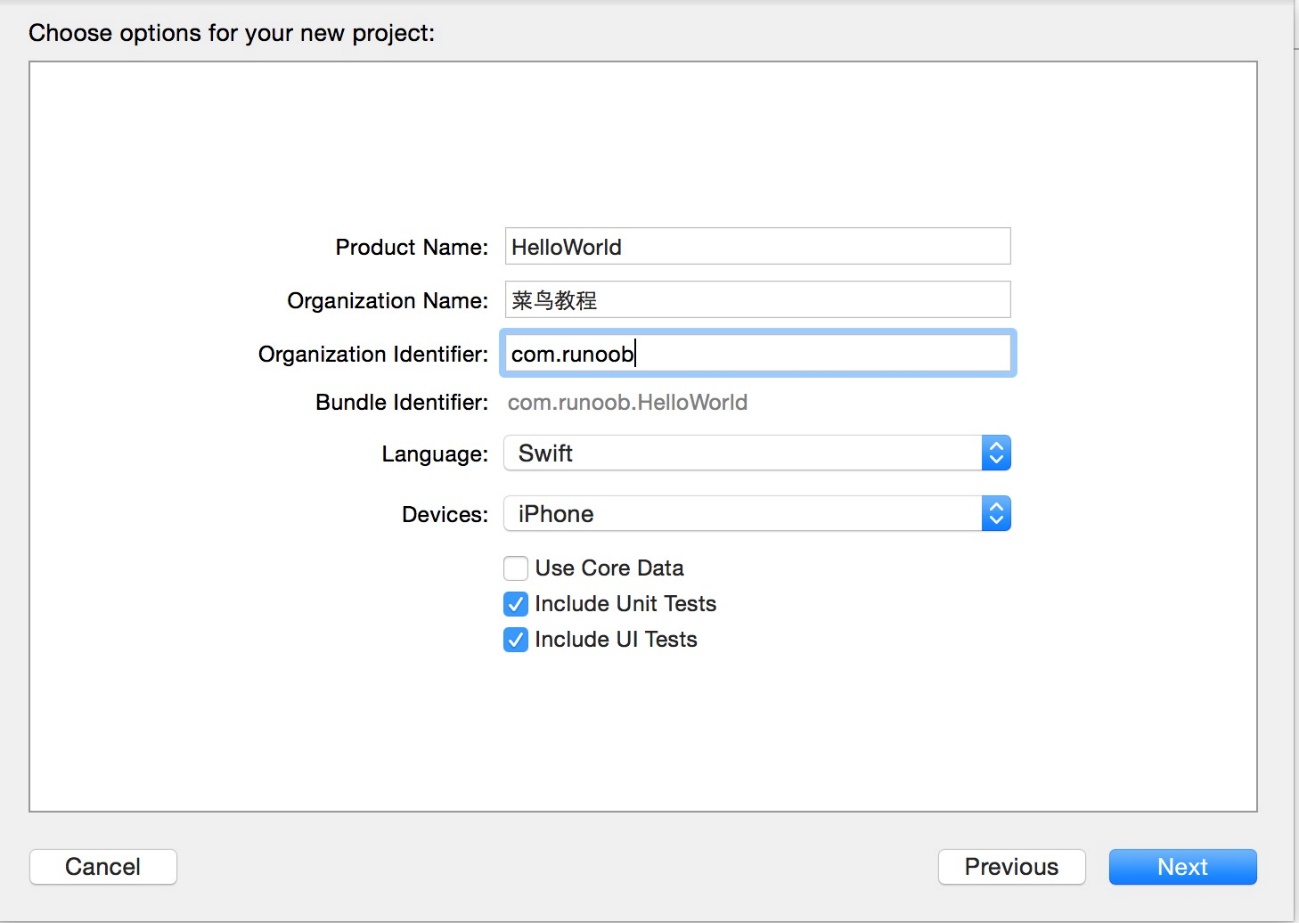


2, nós escolhemos um "Single View Aplicação" e clique em "next", para criar um aplicativo de exemplo aplicativo simples.

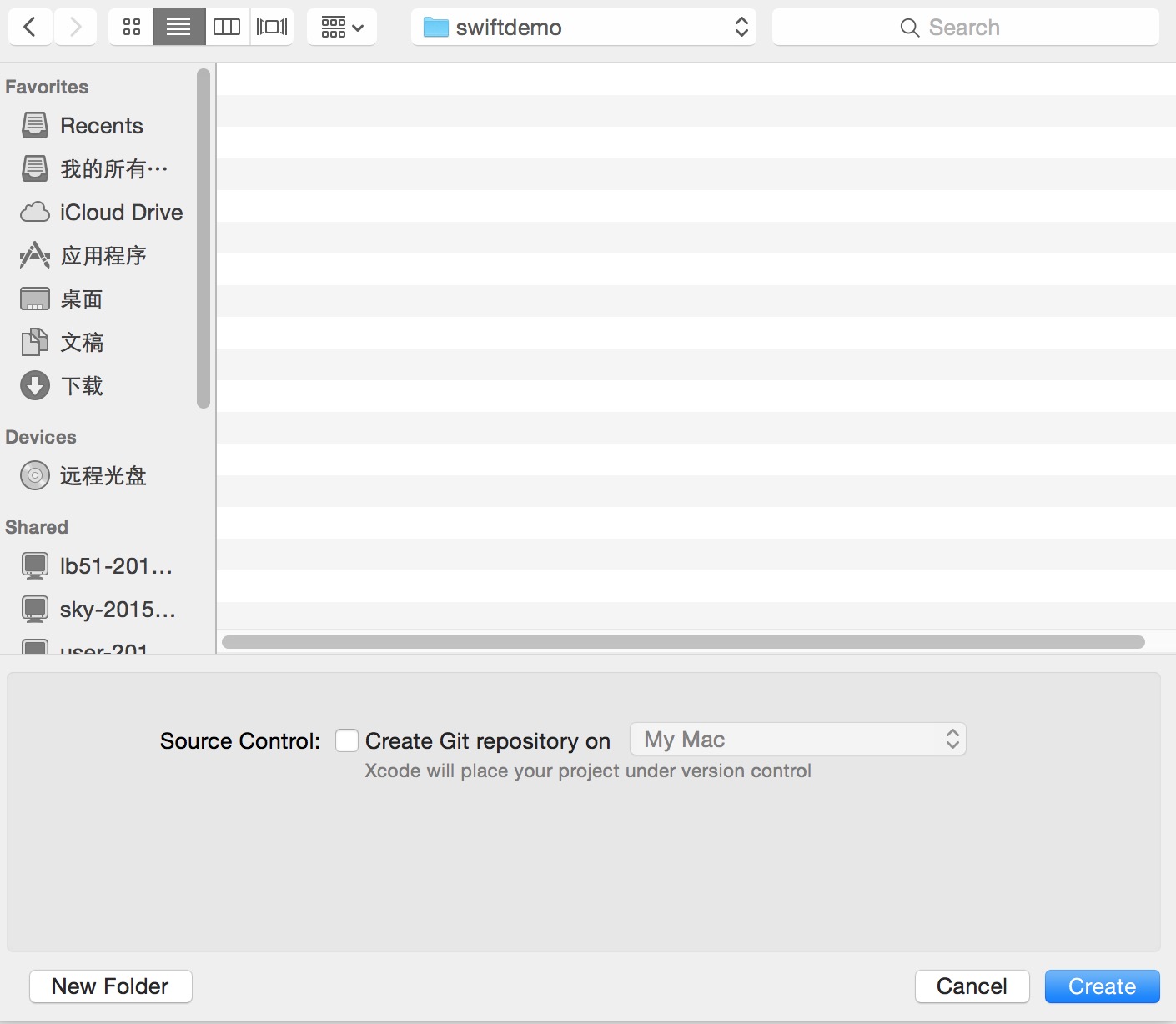


3. Defina suas configurações como: nome do projeto, identificação da organização, tipos de dispositivos e afins conforme mostra a imagem abaixo.

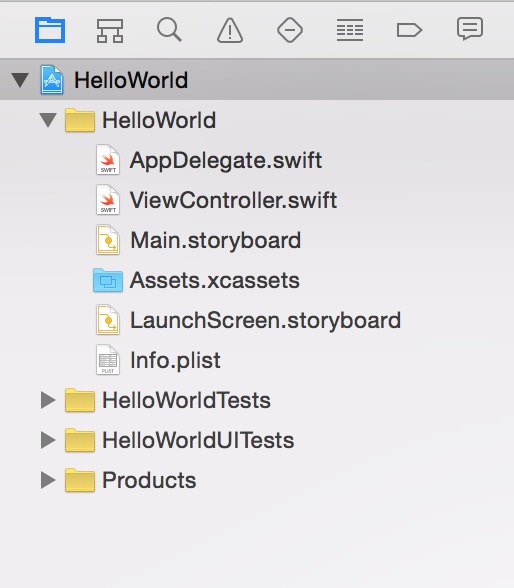
Língua em que existem duas opções: Objective-C e Swift, apenas lembrando que trabalharemos com a linguagem Swift que veio para substituir o Object-C. Escolha o item e clique em "Next" na próxima etapa.



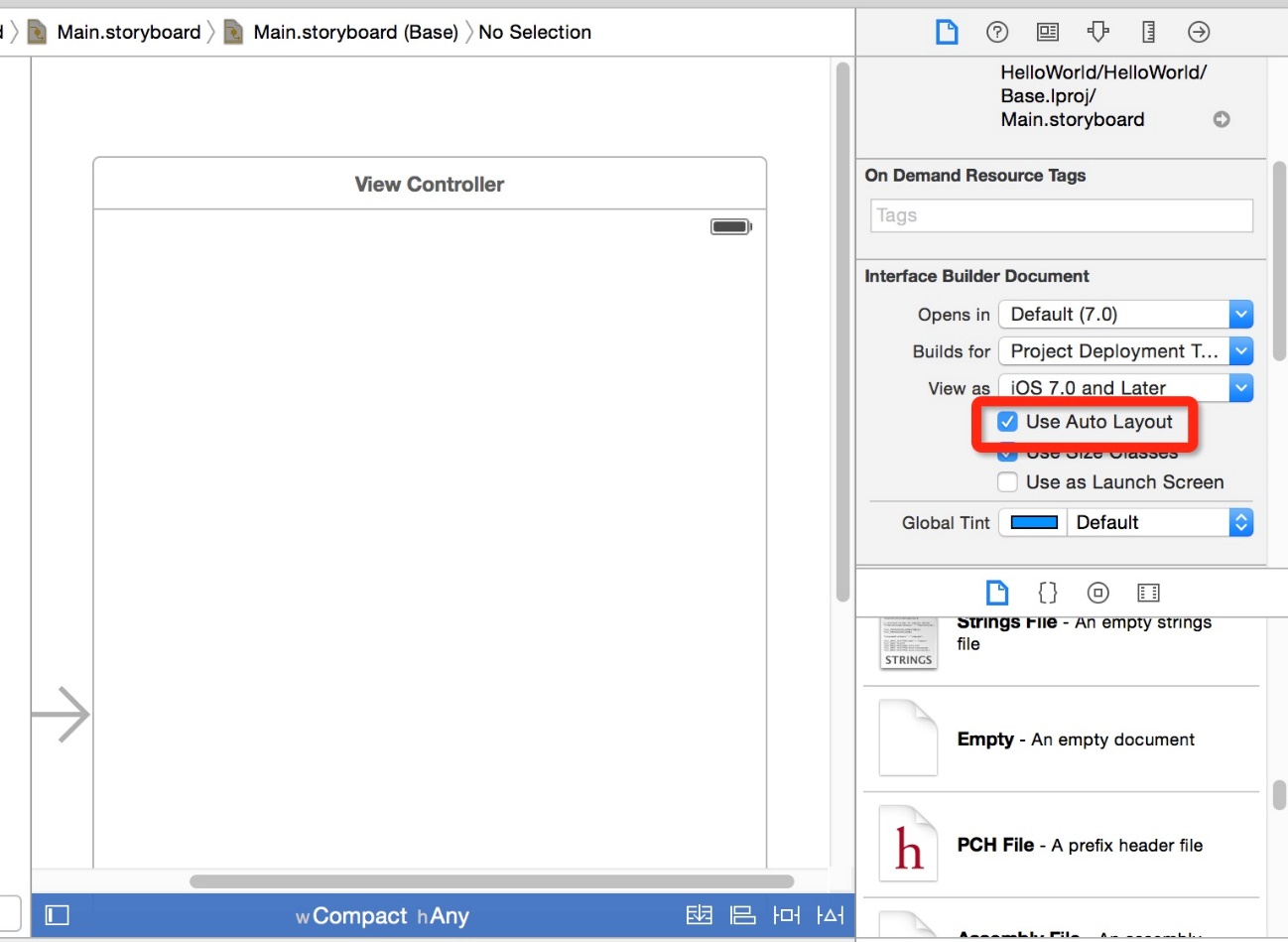
4, selecione o diretório de armazenamento, lembrando que você tem como opção salvar o arquivo e depois copia-lo para passa local do computador físico já que estamos usando a MV para desenvolver.



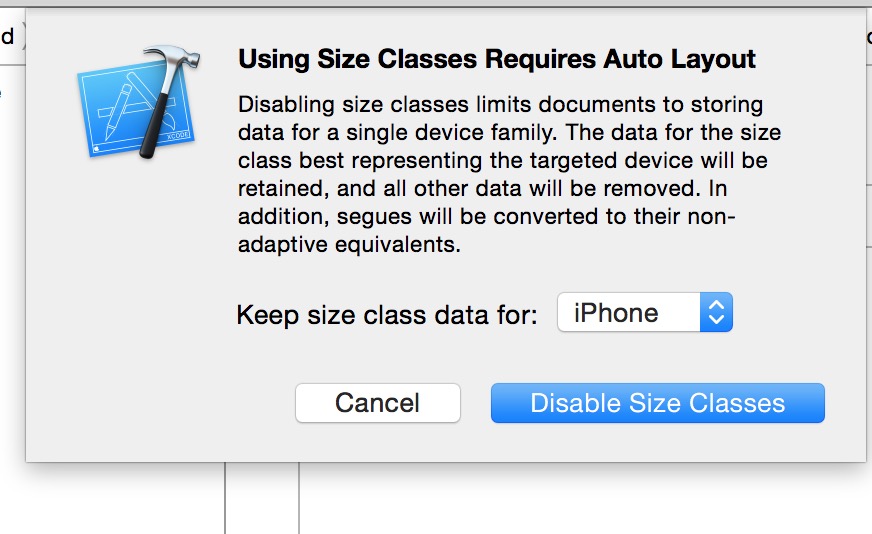
5. Depois que o projeto é criado, o padrão gerar uma estrutura de arquivo como o do exemplo. Não se preocupem em entender por agora, teremos aulas especificas para trabalhar com este tipo de projeto mais adiante. Como falei em aula e como já estamos trabalhando, primeiro vamos aprender a base da linguagem no Playground.



6. O arquivo main.storyboard é o padrão para ver uma interface de aplicativo em branco, simples, do tamanho de tela do smartphone. Podemos optar pela opção de usar o “Auto Layout” para facilitar nosso desenvolvimento.



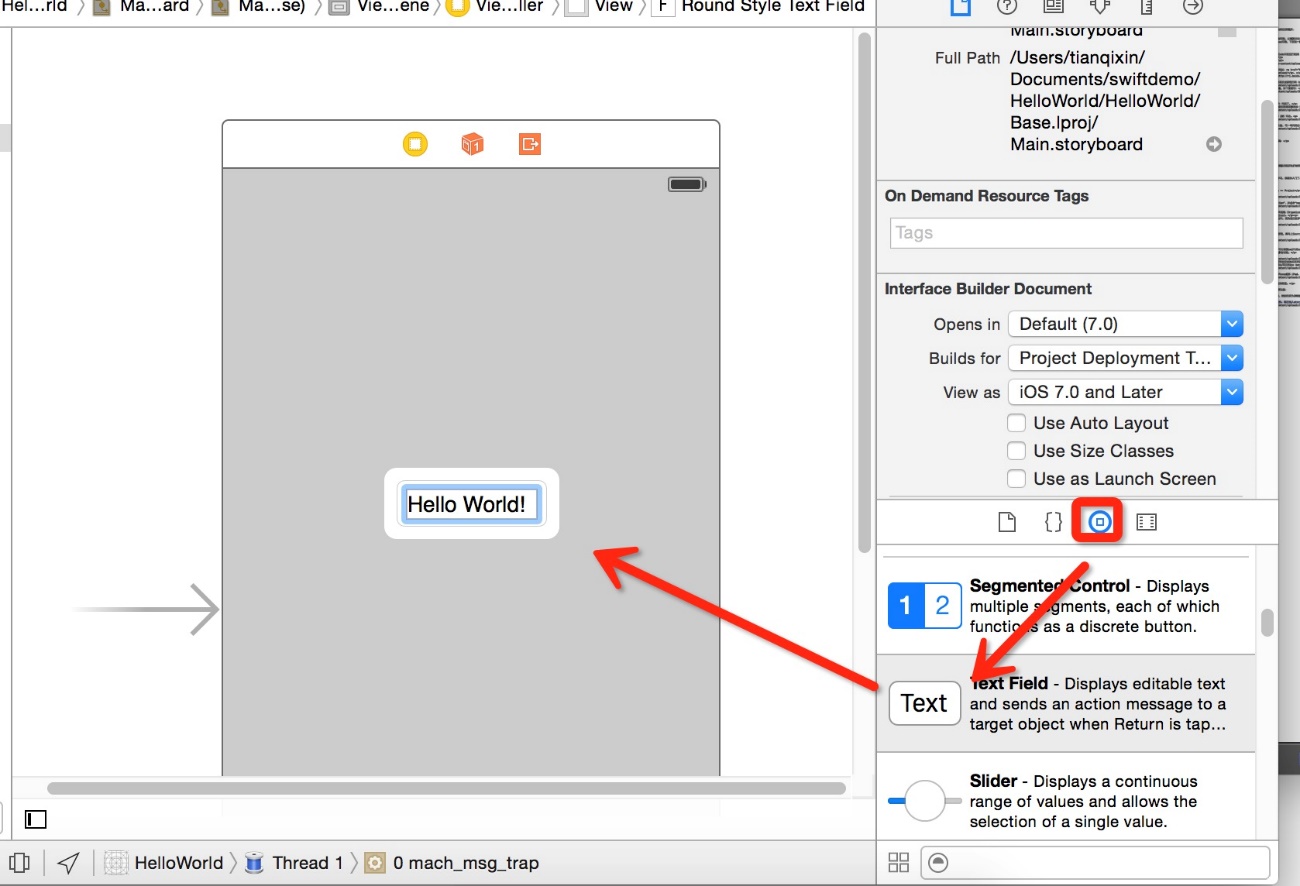
7. Uma caixa de diálogo aparece, vamos escolher o tamanho da interface, iPhone, iPad ou ambos.



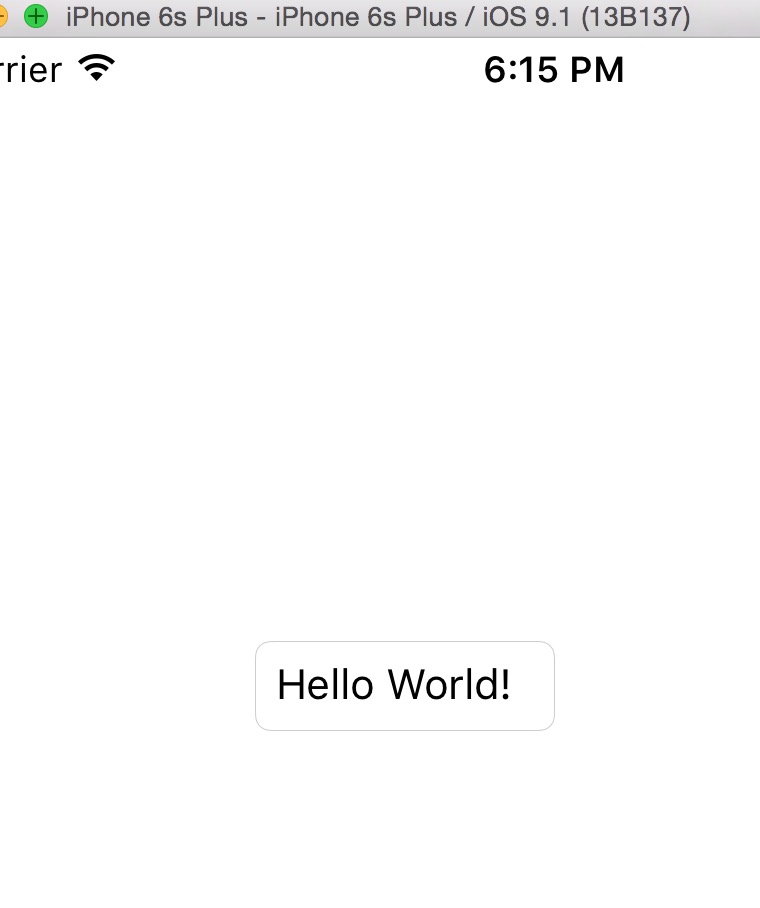
8. Vamos adicionar um conteúdo para fazer uma interface simples.

Abaixo segue um exemplo de como inserir itens na nossa tela e como executar o projeto.

No canto inferior direito do controle, temos algumas ferramentas, nele há um item chamado “texto” conforme o exemplo abaixo. Ele será arrastado para o storyboard, clique duas vezes para escrever o texto "Olá mundo!".



Execute o simulador (atalho Command + R ou escolha do produto => Executar na barra de menu).



Neste ponto, o nosso primeiro projeto Swift está completa.

Mas vamos agora entender um pouco como funcionam as coisas na sintaxe desta nova linguagem que estamos aprendendo.

Abaixo e durante as aulas podemos ver e aplicar os conceitos básicos de programação dentro da linguagem Swift.

Assim vamos avançando e dando sequencia nos nossos projetos.

VOLTE AO PLAYGROUND PARA PRATICARMOS AS QUESTOES ABAIXO E POSTERIORMENTE LEVAMOS ISSO PARA NOSSAS APLICAÇÕES.

# A sintaxe básica:

# Comentarios:

Os comentários de Swift, assim com a linguagem C são semelhante ao de uma única linha comentários começam com duas barras invertidas:

//IComentario de linha

comentários de várias linhas começam com / \* a \* / fim:

/\* Comentario de bloco

Linha 01

Linha 02 \*/

# Ponto e virgula

Ao contrário de outras línguas são, Swift não requer o uso de um ponto e vírgula no final de cada linha da demonstração (;), mas devem ser separados por um ponto e vírgula no mesmo número de linhas declaração escrita:

import UIKit

/\* Swift \*/

var minhavariavel = "Hello, World!"; print(myString)

# Identificadores

Identificador é dar variáveis, constantes, métodos, funções, enumeração, que especifica o nome da estrutura, da classe, e outros protocolos. Palavras que constituem identificadores tem certas normas.

Nomeando identificadores de idioma Swift como segue:

* Maiúsculas e minúsculas, Myname com myname são dois identificadores diferentes;
* Os identificadores podem começar com \_(underline) ou letras, mas não pode ser um número;

Por exemplo: username, User\_Name, \_sys\_val, altura e outros.

# Palavra chave

Uma palavra-chave é uma sequência de carácter reservado semelhante a um identificador, a menos que você utilize acentos ( ` ~ ^ ) elas não podem ser utilizadas como identificadores. Palavras-chave são tem um significado especial pré-definido para o compilador.

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASS** | **DEINIT** | **ENUM** | **EXTENSION** |
| **FUNC** | **IMPORT** | **INIT** | **INTERNAL** |
| **LET** | **OPERATOR** | **PRIVATE** | **PROTOCOL** |
| **PUBLIC** | **STATIC** | **STRUCT** | **SUBSCRIPT** |
| **TYPEALIAS** | **VAR** | **PRECEDENCE** | **RIGHT** |
| **AS** | **DYNAMICTYPE** | **FALSE** | **IS** |
| **NIL** | **SELF** | **SELF** | **SUPER** |
| **TRUE** | **\_COLUMN\_** | **\_FILE\_** | **\_FUNCTION\_** |
| **\_LINE\_** | **WEAK** | **WHILE** | **BREAK** |
| **CASE** | **CONTINUE** | **DEFAULT** | **DO** |
| **ELSE** | **FALLTHROUGH** | **FOR** | **IF** |
| **IN** | **RETURN** | **SWITCH** | **WHERE** |
| **CONVENIENCE** | **DYNAMIC** | **DIDSET** | **ASSOCIATIVITY** |
| **GET** | **INFIX** | **INOUT** | **FINAL** |
| **LEFT** | **MUTATING** | **NONE** | **LAZY** |
| **OPTIONAL** | **OVERRIDE** | **POSTFIX** | **NONMUTATING** |
| **PREFIX** | **PROTOCOL** | **REQUIRED** | **WILLSET** |
| **SET** | **TYPE** | **UNOWNED** |  |

# espaço Swift

Em Swift o uso de espaço não é como em C / C ++, Java, que ignora completamente espaços. A utilização dos espaços de Swift tem certos requisitos.

Em Swift, o operador não pode ser seguido imediatamente em uma variável ou constante. Por exemplo, será dado o seguinte código:

let a= 1 + 2

A mensagem de erro é:

error: prefix/postfix '=' is reserved

O código a seguir está sendo dado (observe o espaço continuação):

let a = 1+ 2

A mensagem de erro é:

error: consecutive statements on a line must be separated by ';'

Só desta forma não vai ocorrer erros:

let a = 1 + 2; // Desta maneira, tudo separado é permitido.

let b = 3+4 // Ou desta maneira, tudo junto também é.

# Tipos de Dados em Swift

Quando usamos qualquer tipo de linguagem de programação, você precisa usar uma variedade de tipos de dados para armazenar informações diferentes.

O tipo de dados da variável determina os valores na memória do computador. Você também pode especificar o tipo de dados quando você declarar uma variável.

Todas as variáveis ​​têm o tipo de dados para determinar quais os dados podem ser armazenados.**Tipos de dados**

Swift fornece uma rica tipos de dados, aqui estão vários tipos de dados comumente usados:

### Int

Geralmente, você não precisa designado especificamente inteiros de comprimento. Swift fornece um tipo especial inteiro Int , o comprimento do vocabulário original e o mesmo comprimento de outras plataforma:

Int tem um intervalo inteiro que pode ser alcançado

 -2,147,483,648 ~ 2,147,483,647 , a maior parte do tempo, esta é já suficientemente grande.

### Uint

Swift também oferece um tipo sem sinal especial UInt , o comprimento do vocabulário original e o mesmo comprimento da plataforma de corrente:

Nota:   
Tente não usar UInt , a menos que você realmente precisa para armazenar um valor inteiro sem sinal. Mesmo assim, é preferível utilizar Int , mesmo se quiser armazenar um valor não-negativo.

### Flutuador

Float é um número com uma parte decimal, como 3.14159 , 0.1 e -273.15 .

Mais ampla do que a representação tipo de ponto flutuante do tipo inteiro que pode armazenar mais de Int tipo números maiores ou menores. Swift tem dois tipos de ponto flutuante:

* Double representam ponto flutuante de 64 bits. Quando você precisa para armazenar grande número ou muito alta precisão de ponto flutuante, use este tipo.
* Float representa o número de ponto flutuante de 32 bits. Menos de precisão, então podemos usar este tipo.

Nota:   
Double precisão é muito elevado, pelo menos 15 dígitos, Float aconteceu apenas seis dígitos. Selecione o intervalo de valores dependendo do tipo precisam ser abordadas seu código.

### Valor Booleano

Swift tem um tipo booleano básica (Boolean), chamado Bool. valor booleano refere-se à lógica de valor, porque eles só podem ser verdadeiras ou falsas. Swift tem dois constante booleana, o verdadeiro e o falso.

### String

String é um conjunto de sequências de caracteres, tais como:

"Hello, World!"

### Char

Caráter refere-se a uma única letra, por exemplo:

"C"

### tipo opcional

Use o tipo de opcional (opcionais) para lidar com o caso os valores em falta. opcional tipo representa um valor ou nenhum valor.

# faixa de valores

A tabela a seguir mostra o espaço mínimo e máximo de armazenamento de diferentes tipos de variáveis ​​de memória e tipos de variáveis:

|  |  |  |
| --- | --- | --- |
| **tipo** | **Tamanho (bytes)** | **Valor intervalo** |
| iNT8 | 1 byte | -127 A 127 |
| uint8 | 1 byte | 0-255 |
| Int32 | 4 bytes | -2147483648 A 2147483647 |
| UInt32 | 4 bytes | 0-4294967295 |
| Int64 | 8 bytes | -9223372036854775808 Para 9223372036854775807 |
| UInt64 | 8 bytes | 0-18446744073709551615 |
| flutuador | 4 bytes | 1.2E-38 a 3.4E + 38 (~ 6 dígitos) |
| duplo | 8 bytes | 2.3E-308 a 1.7E + 308 (~ 15 dígitos) |

## **Alias de tipo**

Digite um alias para o tipo atual define um outro nome, digite o alias definido usando typealias palavra-chave. Sintaxe é como se segue:

typealias newname = type

O exemplo a seguir define os pés tipo de alias Int:

typealias Feet = Int

Agora, podemos definir um alias de variável:

import Cocoa

typealias Feet = Int

var distance: Feet = 100

print(distance)

Nós usamos o playground o programa acima, a saída é:

100

# Tipo de segurança

Swift é um tipo de linguagem seguro.

Por Swift ser de tipo seguro, ele vai inspeção (verificações de tipo) o tipo das variáveis antes de compilar o código, se não combinam com o tipo demarcado ele repassara um erro. Isto permite-lhe encontrar e corrigir erros no início do desenvolvimento.

import UIKit

var varA = 42

varA = "This is hello"

print(varA)

O procedimento acima será dada no Xcode:

error: cannot assign value of type 'String' to type 'Int'

varA = "This is hello"

Significado não pode ser string 'String' atribuído à variável 'int'.

Inferência de tipos

Quando você tem que lidar com diferentes tipos de valores, a verificação de tipo pode ajudar a evitar erros. No entanto, isso não significa que toda vez que você declarar constantes e variáveis ​​precisa especificar explicitamente o tipo.

Se você não especificar explicitamente o tipo, Swift usa o tipo de inferência (inferência de tipos) para selecionar o tipo apropriado.

Por exemplo, se você der um novo valor constante 42 e não indica o tipo, a Swift pode inferir tipo Int constante, porque o valor inicial que você atribua a ele se parece com um inteiro:

let meaningOfLife = 42

// meaningOfLife sera declarado como um Int.

Da mesma forma, se você não tem que indicar o tipo de literais de ponto flutuante, Swift irá inferir que você quer é duplo:

let pi = 3.14159

// pi será declarado como Double

Quando um tipo de inferência de ponto flutuante, Swift vai sempre escolher em vez de flutuante duplo.

Se a mesma expressão como inteiro e ponto flutuante, serão inferidos tipo Double:

let outroPi = 3 + 0.14159

// outroPi também ficará como um Double

O valor original de 3 de não declarar explicitamente o tipo, e a expressão não é um ponto flutuante literal, a expressão será presumido que tipo Double.

### Exemplos

var varA = 42

print(varA)

var varB = 3.14159

print(varB)

var varC = 3 + 0.14159

print(varC)

Implementação do código acima, a saída é:

42

3.14159

3.14159

# Variáveis em Swift

Variável é um espaço reservado conveniente para endereços de memória do computador de referência.

Em Swift para cada variável é atribuído um tipo específico, que determina o tamanho da memória variável para os diferentes tipos de dados, também pode determinar o intervalo de valores que podem ser armazenados.

O último capítulo temos de introduzir os tipos de dados básicos , incluindo Int, Float e Double, tipo booleana e tipo de cadeia de caracteres. Além disso, Swift também fornece outro, mais poderoso tipo de dados, opcional, Array, dicionário, Struct, Classe, e similares.

Em seguida, vamos discutir a forma de declarar o programa Swift e você usar variáveis.

# Declaração da variável

Declarações de variáveis ​​significa dizer ao compilador para criar mais espaço de armazenamento na memória que posição sera uma variável.

Antes de usar uma variável, você precisa usar a palavra-chave **var declará-la da seguinte forma:**

var nomeVariavel = <valor inicial>

O que se segue é um exemplo simples de uma declaração de variável programa Swift:

import UIKit

var varA = 42

print(varA)

var varB: Float

varB = 3.14159

print(varB)

Os resultados da execução do programa acima:

42

3.14159

Ou seja, você pode optar por declarar as variáveis implícita ou explicitamente conforme exemplos acima.

# Nomenclatura de variáveis

Os nomes das variáveis ​​podem consistir de letras, números e sublinhados.

Os nomes das variáveis ​​precisam começar com uma letra ou um sublinhado.

Swift é uma linguagem case-sensitive, por isso, letras maiúsculas e minúsculas não são os mesmos.

# Variável de saída

Variáveis e constantes podem usar **print (swift 2 substitui o println de impressão) para a saída.**

Parênteses podem ser usados ​​na seqüência com uma barra invertida para inserir variáveis ​​seguintes exemplos:

import UIKit

var name = "Pedro"

var site = "http://www.w3big.com"

print("\(name) POSSUI O SITE：\(site)")

# Variável de Tipos Opcionais

Opcional é um tipo variável usado para manusear os valores em falta. Meios opcionais "onde há um valor, e é igual a x" ou "não há nenhum valor."

Opcional é como um tipo chamado de taquigrafia, em outras palavras, as duas instruções a seguir são equivalentes:

var optionalInteger: Int?

var optionalInteger: Optional<Int>

Em ambos os casos, o optionalInteger são variáveis do tipo inteiro opcional. Note-se que, no tipo de declaração com ?(interrogação) não há espaço entre.

Opcional é usado para indicar que pode ou não ter valor na variavel. Qualquer tipo pode ser declarado explicitamente como (ou conversão implícita) tipo opcional. Quando você declarar um tipo opcional, certifique-se de usar os suportes (?).

Quando você declara um atributo, variável ​​ou opcionais, não está disponível quando o valor inicial, pois seu valor padrão é nulo.

Usar o operador (!) para adquirir uma variável facultativa é nula haverá erros de tempo de execução.

Você pode usar um link opcional e um ligamento opcional seletivamente para executar operações sobre a expressão opcional. Se o valor for nulo, nenhuma operação for implementada, não vai ser executado erro.

Vamos dar uma olhada nos seguintes exemplos para aprender tipos Swift opcionais de aplicações:

import UIKit

var minhaString:String? = nil

if minhaString!= nil {

print(minhaString)

}else{

print("valor dela")

}

## **Forçado a resolver**

Depois de determinar o tipo contém valores opcionais, você pode adicionar um nome opcional por trás de um ponto de exclamação (!). Para obter o valor use o ponto de exclamação, ele mais ou menos “diz”: "Eu sei que existem valores opcionais, eu garanto, pode usá-lo." Isso é chamado obrigatória valor analisado opcional (forçado desembrulhar).

Nota:   
Usar ! para obter um valores opcionais inexistentes irá resultar em um erro de execução. Antes de forçar o valor analítico, você deve determinar que variavel não contem valor nil(nulo).

**Analisar automaticamente**

Você pode substituir um ponto de interrogação quando você declarar um ponto de exclamação opcional variável (!) (?). Ao usar essas variáveis ​​opcionais que você não precisa mais um ponto de exclamação (!) Para obter o valor, ele irá analisar automaticamente.

import UIkit

var myString:String!

myString = "Hello, Swift!"

}

# Bind opcional

Usando as ligações opcionais (opcional de ligação) para determinar se um tipo de opcionais contém o valor, se ele contém um valor atribuído para colocar uma variável temporária ou constante. Opcional de ligação podem ser utilizados em se e enquanto demonstrações para avaliar o valor de um tipo opcional e atribuir um valor de uma constante ou variável.

Como isso se declaração por escrito em um opcional de ligação:

if let constantName = someOptional {

statements

}

Vejamos um exemplo de ligação alternativa simples:

var myString:String?

myString = "Hello, Swift!"

if let yourString = myString {

print("A variavel contem - \(yourString)")

}else{

print("Valor nulo")

}

# Constantes

Uma vez definido, o programa está sendo executado não é possível alterar o seu valor.

Constantes podem ser de qualquer tipo de dados, tais como: integer, ponto flutuante, caracteres ou boolean. Há também são enumerados constantes do tipo:

Constantes são como variáveis ​​exceto pelo fato que o valor de uma constante não pode ser alterado uma vez definida, e o valor de uma variável pode ser alterada.

# Declaração constante

Constantes são declaradas usando a palavra-chave **let.**

let constantName = <valor>

O que se segue é um exemplo simples de constantes do programa Swift:

let constA = 42

print(constA)

# Operadores Swift

Um operador é um símbolo, usado para dizer ao compilador para executar as operações matemáticas ou lógicas.

Swift fornece os seguintes operadores:

* operadores aritméticos
* comparação
* Operadores lógicos
* Operadores de atribuição
* Operadores de intervalo
* outros operadores

Neste capítulo, vamos explicar-lhe operadores aritméticos, operadores relacionais, operadores lógicos, operadores bit a bit, operador de atribuição e outros operadores.

# Operadores aritméticos

A tabela a seguir lista os operadores aritméticas suportados pela língua:

|  |  |  |
| --- | --- | --- |
| **operadores** | **descrição** | **Exemplos** |
| + | mais | Resultados para A + B 30 |
| - | menos | A - B -10 resultados |
| \* | sinal de multiplicação | A resultados \* B para 200 |
| / | Sinal de divisão | B / A 2 resultados |
| % | restante | B% Um resultado é 0 |
| ++ | incremento | A ++ 11 resultados |
| - | diminuição | resultado A-- 9 |

### Exemplos

A seguir é um exemplo simples de operações aritméticas:

var A = 10

var B = 20

print("A + B é igual：\(A + B)")

print("A - B é igual：\(A - B)")

print("A \* B é igual：\(A \* B)")

print("B / A é igual：\(B / A)")

A++

print("A++ depois A é igual \(A)")

B--

print("B-- depois B é igual \(B)")

Os resultados da execução do programa acima:

A + B é igual：30

A - B é igual：-10

A \* B é igual：200

B / A é igual：2

A++ depois A é igual 11

B-- depois B é igual 19

# Comparação

A tabela a seguir lista os operadores de comparação suporte ao idioma Swift, em que as variáveis ​​A 10, B 20 variáveis:

|  |  |  |
| --- | --- | --- |
| **operadores** | **descrição** | **Exemplos** |
| == | igual | (A == B) é falsa. |
| ! = | não é igual | (A! = B) é verdadeiro. |
| > | maior do que | (A> B) é falsa. |
| < | menos que | (A <B) é verdadeiro. |
| > = | Maior do que ou igual | (A> = B) é falsa. |
| <= | Menor ou igual | (A <= B) é verdadeiro. |

### Exemplos

A seguir é um exemplo simples de operações de comparação:

var A = 10

var B = 20

print("A == B efeito：\(A == B)")

print("A != B efeito：\(A != B)")

print("A > B efeito：\(A > B)")

print("A < B efeito：\(A < B)")

print("A >= B efeito：\(A >= B)")

print("A <= B efeito：\(A <= B)")

Os resultados da execução do programa acima:

A == B efeito：false

A != B efeito：true

A > B efeito：false

A < B efeito：true

A >= B efeito：false

A <= B efeito：true

# Operadores lógicos

A tabela a seguir lista os operadores lógicos de suporte ao idioma Swift, onde a variável A é verdade, a variável B é falso:

|  |  |  |
| --- | --- | --- |
| **operadores** | **descrição** | **Exemplos** |
| && | Lógica e. Se o operador for TRUE ambos os lados era verdade. | (A && B) é falsa. |
| || | Ou lógica. Se o operador em ambos os lados de pelo menos um dos verdadeiros era verdade. | (A || B) é verdadeiro. |
| ! | NÃO lógico. Ele inverte o valor booleano, tão verdadeiro se torna falsa, falsa se torna verdade. | ! (A && B) é verdadeiro. |

A seguir é um exemplo simples da operação lógica:

var A = true

var B = false

print("A && B Efeito：\(A && B)")

print("A || B Efeito：\(A || B)")

print("!A Efeito：\(!A)")

print("!B Efeito：\(!B)")

Os resultados da execução do programa acima:

A && B Efeito：false

A || B Efeito：true

!A Efeito：false

!B Efeito：true

# Operadores de atribuição

A tabela a seguir lista o idioma Swift operador básico de atribuição:

|  |  |  |
| --- | --- | --- |
| **operadores** | **descrição** | **Exemplos** |
| = | Simples operador de atribuição, especifica o operando direito atribuído ao operando esquerdo. | C resultado = A + B + A operação B é atribuído a C |
| + = | Adicionando após a atribuição, os lados esquerdo e direito do operando antes de adicionar atribuído ao operando esquerdo. | C + = A corresponde a C = C + A |
| - = | Depois de atribuição de subtração, os operandos esquerdo e direito de ambos os lados do operando esquerdo após subtracção atribuído. | C - = A corresponde a C = C - A |
| \* = | Em seguida, multiplicando-se a atribuição, os lados esquerdo e direito dos operandos são multiplicados antes atribuição para o operando esquerdo. | C \* = A corresponde a C = C \* A |
| / = | Dividido após a atribuição, os lados esquerdo e direito do operando divisória após atribuição para o operando esquerdo. | C / = A é equivalente C = C / A |
| % = | Remanescente após a atribuição, os lados esquerdo e direito do operando resto depois de atribuição para o operando esquerdo. | C% = A é equivalente a C = C% A |

A seguir é um exemplo simples do operador de atribuição:

var A = 10

var B = 20

var C = 100

C = A + B

print("C Efeito：\(C)")

C += A

print("C Efeito：\(C)")

C -= A

print("C Efeito：\(C)")

C \*= A

print("C Efeito：\(C)")

C /= A

print("C Efeito：\(C)")

C %= A

print("C Efeito：\(C)")

Os resultados da execução do programa acima:

C Efeito：30

C Efeito：40

C Efeito：30

C Efeito：300

C Efeito：30

# Operadores de intervalo

Swift oferece dois operadores de intervalo.

|  |  |  |
| --- | --- | --- |
| **operadores** | **descrição** | **Exemplos** |
| Operadores intervalo fechado | operador intervalo fechado (a ... b) um intervalo definido contém todos os valores de a para b (incluindo a e b), b deve ser maior do que ou igual a. Fechado operador iteração intervalo numa gama de valores de todos é muito útil, como é o para-em malha: | 1 ... 5 gama é 1, 2, 3, 4 e 5 |
| operador de intervalo semi-aberto | Semi-aberto intervalo (a ..<="" td="" style="box-sizing: inherit;"> | 1 .. <5 valor do intervalo de 1, 2, 3, e 4 |

A seguir é um exemplo simples de aritmética intervalar:

for index in 1...5 {

print("\(index) \* 5 = \(index \* 5)")

}

for index in 1..<5 {

print("\(index) \* 5 = \(index \* 5)")

}

Os resultados da execução do programa acima:

1 \* 5 = 5

2 \* 5 = 10

3 \* 5 = 15

4 \* 5 = 20

5 \* 5 = 25

1 \* 5 = 5

2 \* 5 = 10

3 \* 5 = 15

4 \* 5 = 20

# Outros operadores

Swift oferece outros tipos de operadores, tais como um o operador ternário.

* Três operador ternário operar o destino da operação e linguagens C, Swift é apenas uma operador ternário, é operador ternário ( a ? b : c ).

|  |  |  |
| --- | --- | --- |
| **operadores** | **descrição** | **Exemplos** |
| Unario | Adicionar uma frente digitais - Prefixo | -3 Ou -4 |
| Binario | Dinheiro Digital Adicionar + Prefixo | 6 Resultado 6 |
| Operador ternário | ? Condição X: Y | Se retornar true, adiciona o valor de X, caso contrário, Y |

O que se segue é um exemplo simples de operações unários, binários e ternários:

var A = 1

var B = 2

var C = true

var D = false

print("-A é igual：\(-A)")

print("A + B é igual：\(A + B)")

print("Operação Ternaria：\(C ? A : B )")

print("Operação Ternaria：\(D ? A : B )")

Os resultados da execução do programa acima:

-A é igual：-1

A + B é igual：3

Operação Ternaria：1

Operação Ternaria：2

# Declaração condicional

Instruções condicionais através de uma ou mais condições previstas para executar o programa, execute a condição especificada afirmação como verdadeira, execute condição adicional de instrução especificada como falsa.

Swift fornece os seguintes tipos de instruções condicionais:

|  |  |
| --- | --- |
| **demonstrações** | **descrição** |
| if | **if consiste de uma expressão booleana e uma ou mais declarações de execução.** |
| if ... else | Você pode ter uma **outra declaração opcional if, else executa a expressão booleana é falsa.** |
| if ... else if ... else | Você pode ter um opcional **else if após if...else, else if ... else é muitas vezes usado para determinar uma série de condições.** |
| switch | switch permite que uma variável igual a uma pluralidade de valor de teste. |

## **gramática**

**a sintaxe da instrução:**

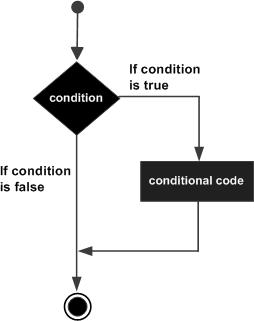
if ExpressaoBoleana {

/\* Codigo dos acontecimentos aqui \*/

}

Se a expressão booleana for **verdadeira, a instrução if dentro de um bloco de código será executado.** Se a expressão booleana é **falsa, se o primeiro conjunto de códigos após a declaração (após o fechamento parênteses) será executada.**

## **fluxograma**



## **Exemplos**

var varA:Int = 10;

if varA < 20 {

print("varA menor que 20");

}

print("varA é igual \(varA)");

Quando o código acima é compilado e executado, ele produz os seguintes resultados:

varA menor que 20

varA é igual 10

# Swift if ... else

Depois de um **if com uma instrução else opcional, else executa a expressão booleana é falsa.**

## **gramática**

sintaxe do **if ... else:**

if boolean\_expression {

/\* Codigo dos acontecimentos aqui caso seja verdadeiro \*/

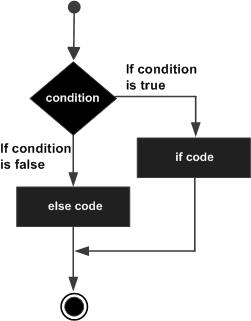
} else {

/\* Codigo dos acontecimentos aqui caso seja falso \*/

}

Se a expressão booleana for **verdade, então o código entrara no primeiro bloco de execução.** Se a expressão booleana é **falsa, então o código entra no bloco else executado.**

## **fluxograma**



## **Exemplos**

var varA:Int = 100;

if varA < 20 {

print("varA é menor que 20");

} else {

print("varA é maior que 20");

}

print("varA tem o valor de \(varA)");

Quando o código acima é compilado executado, ele irá produzir os seguintes resultados:

varA é maior que 20

varA tem o valor de 100

# Swift if ... else if ... else

Depois de um **if com um opcional else if ... else, else if ...else no teste mais de uma instrução de condição é muito útil.**

Quando você usa if, else if, else precisa observar a seguinte declaração:

* Pode ter zero ou um outro lugar depois da instrução if, mas se else if, else else if necessário após o comunicado.
* Pode ter zero ou mais instruções depois da instrução if else if, else if declaração deve aparecer antes do else.
* Uma vez que outra declaração executado com sucesso, o outro, ou então se else não são executados.

**gramática**

if boolean\_expression\_1 {

/\* boolean\_expression\_1 \*/

} else if boolean\_expression\_2 {

/\* boolean\_expression\_2 \*/

} else if boolean\_expression\_3 {

/\* boolean\_expression\_3 \*/

} else {

/\* true \*/

}

### Exemplos

var varA:Int = 100;

if varA == 20 {

print("varA vale 20");

} else if varA == 50 {

print("varA vale 50");

} else {

print("O valor não é 20 nem 50");

}

print("varA é igual \(varA)");

Quando o código acima é compilado executado, ele irá produzir os seguintes resultados:

O valor não é 20 nem 50

varA é igual 100

# If aninhados ou encadeados

Você pode usar outro if ou else if declarado dentro de um if ou else if.

## **gramática**

linguagem Swift aninhados if sintaxe:

if boolean\_expression\_1 {

/\* boolean\_expression\_1 \*/

if boolean\_expression\_2 {

boolean\_expression\_1 \*/

}

}

Você pode aninhar **else if ... else, forma semelhante à *IF* aninhadas.**

### Exemplos

var varA:Int = 100;

var varB:Int = 200;

if varA == 100 {

print("A verificação é true");

if varB == 200 {

print("A verificação é true");

}

}

print("varA vale \(varA)");

print("varB vale \(varB)");

Quando o código acima é compilado executado, ele irá produzir os seguintes resultados:

A verificação é true

A verificação é true

varA vale 100

varB vale 200

# Switch Case

**Switch permite que uma variável seja verificada se é igual a uma pluralidade de valor de teste.** Basta combinar a exposição do caso, a instrução é executada para completar a chave inteira.

## **gramática**

Sintaxe da instrução switch:

switch expressao {

case expressao1:

statement(s)

case expression2:

statement(s)

case expression3:

statement(s)

default : /\* Caso nenhuma condição seja atendida \*/

statement(s);

}

**Nota:** Na maioria das linguagens o interruptor de bloco de instrução é executado no caso de você querer manter-se com ruptura, ou então executar a instrução após o caso a ordem. Já em linguagem Swift, o padrão não é executado, o interruptor irá terminar. Se você quiser fazer a declaração no caso Swift iria continuar a executar após a ordem, precisamos usar declaração fallthrough.

### exemplo 1

Os exemplos a seguir não estará usando declaração fallthrough:

var valor = 10

switch valor {

case 100 :

print( "index vale 100")

case 10,15 :

print( "index vale 10 ou 15")

case 5 :

print( "index vale 5")

default :

print( "nenhum case")

}

Quando o código acima é compilado executado, ele irá produzir os seguintes resultados:

Index vale 10 ou 15

### Exemplo 2

O exemplo a seguir usa declaração fallthrough:

var index = 10

switch index {

case 100 :

print( "index vale 100")

fallthrough

case 10,15 :

print( "index vale 10 ou 15")

fallthrough

case 5 :

print( "index vale 5")

default :

print( "Nenhum case")

}

Quando o código acima é compilado executado, ele irá produzir os seguintes resultados:

index vale 10 ou 15

index vale 5

# Estrutura de Repetição (loops)

Às vezes, pode ser necessário executar repetidamente a mesma parte do código. Em circunstâncias normais, as instruções são executadas sequencialmente: a primeira instrução na função executada primeiro, seguida por uma segunda declaração, e assim por diante.

As linguagens de programação fornecem várias estruturas de controle de caminhos de execução mais complexos.

Loops nos permitem executar repetidamente uma declaração ou grupo de instruções.

# Tipo de loop

linguagem Swift fornece os seguintes tipos de loop:

|  |  |
| --- | --- |
| **Tipo de ciclo** | **descrição** |
| for-in | Atravessando uma coleção de todos os elementos que, por exemplo, vão desde a representação digital de elementos na matriz, uma cadeia de caracteres. |
| for loop | Para repetir uma série de declarações até atingir certas condições forem cumpridas, geralmente após cada ciclo é completado pelo aumento do valor do contador de alcançar. |
| while loop | Uma série de instruções a serem executadas se a condição for verdadeira, será executado repetidamente até que a condição se torna falsa. |
| repeat ... while loop | Enquanto uma instrução semelhante antes que a diferença é que a condição de ciclo é determinado, o primeiro bloco é executado o primeiro ciclo. |

# Instruções de controle de loop

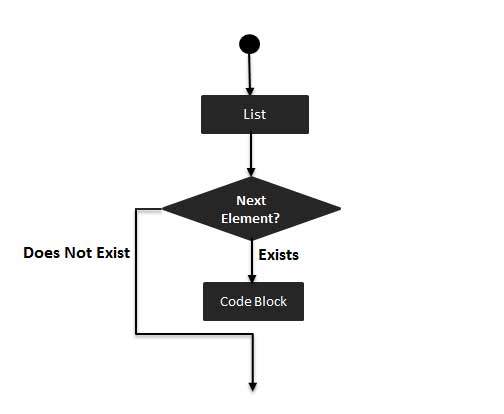
instruções de controle de loop podem alterar a ordem de execução de seu código, através do qual você pode saltar código. Instruções de controle seguinte loop:

|  |  |
| --- | --- |
| **instruções de controle** | **descrição** |
| CONTINUE | Diga uma iteração do loop parar imediatamente este ciclo e começar de novo a próxima iteração. |
| BREAK | Interromper o ciclo atual. |
| FALLTHROUGH | Se um caso após a implementação, continuar com o caso seguinte, que exige o uso de fallthrough (através) palavra-chave. |

Swift para-in loop for para percorrer uma coleção de todos os elementos que, por exemplo, vão desde a representação digital de elementos na matriz, uma cadeia de caracteres.

### For-In

**fluxograma:**



### Exemplos

var meuArray:[Int] = [10, 20, 30]

for index in meuArray {

print( "index vale \(index)")

}

A saída de execução de programa acima é:

index vale 10

index vale 20

index vale 30

# FOR LOOP

O laço é utilizado para repetir uma série de instruções até que certas condições são atingidos, em geral depois de cada ciclo é completado pelo aumento do valor do contador de alcançar.

gramática

Swift para a sintaxe de loop é a seguinte:

for init; condition; increment{

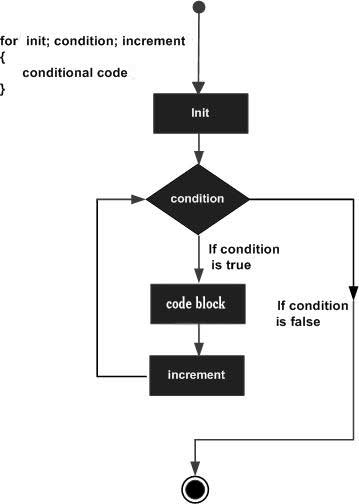
Codigos

}

**parâmetros analíticos:**

1. **init é executado pela primeira vez, e apenas uma vez.** Esta etapa permite que você declare e inicialize todas as variáveis ​​de controle de loop. Você também pode escrever qualquer declaração que não está aqui, contanto que haja um ponto e vírgula.
2. Em seguida, vamos julgar a **condição.** Se for verdadeiro, o corpo do laço é executado. Se for falso, o corpo do laço não é executado, e o fluxo de controle salta imediatamente para o próximo ciclo comunicado.
3. Depois de executar a para o corpo loop, controle de fluxo salta para trás acima de declaração **de incremento.** Esta declaração permite que você atualize a variável de controle loop. A instrução pode ser deixada em branco, enquanto as condições parecem ter um ponto e vírgula.
4. Condição é julgado novamente. Se for verdade, então o ciclo de execução, o processo é repetido (corpo do laço, e em seguida, aumentar o valor do passo, em seguida, re-determinar as condições). Quando a condição se torna falsa, por loop termina.

**fluxograma:**



Exemplos

var meuArray:[Int] = [10, 20, 30]

for var index = 0; index < 3; ++index {

print( “Posicao [\(index)] contem o valor \( meuArray [index])")

}

A saída de execução de programa acima é:

Posicao [0] contem o valor 10

Posicao [1] contem o valor 20

Posicao [2] contem o valor 30

# While Loop

O while loop desde o início testa uma única condição. Se a condição for verdadeira, executa uma série de declarações é repetido até que a condição se torna falsa.

### gramática

Swift sintaxe loop while é a seguinte:

while condition

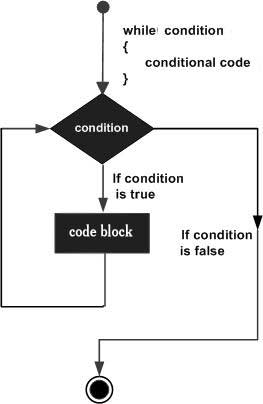
{

Codigos

}

Se a condição for verdadeira, executa uma série de declarações é repetido até que a condição se torna falsa.

**fluxograma:**



### Exemplos

var index = 10

while index < 20

{

print( "index contem o valor \(index)")

index = index + 1

}

A saída de execução de programa acima é:

index contem o valor 10

index 的contem o valor 11

index contem o valor 12

index contem o valor 13

index contem o valor 14

index contem o valor 15

index contem o valor 16

index contem o valor 17

index contem o valor 18

index contem o valor 19

# Repeat-While

Enquanto um loop como o for e while, antes do laço começar ja determinam a instrução condicional, o repeat determina no final do ciclo a condição de execução.

### gramática

repeat Swift ... enquanto sintaxe ciclo é o seguinte:

repeat

{

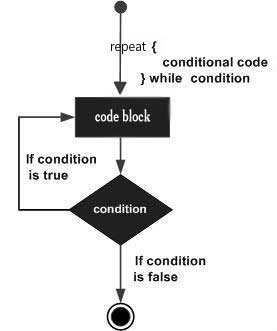
statement(s);

}while( condition );

Note que as expressões condicionais aparecem no final do ciclo, de modo que o comando (s) circuito será executada pelo menos uma vez antes que a condição é testada.

Se a condição for verdadeira, o fluxo de controle salta para o topo da repetição, em seguida, repete a declaração do(s) loop(s). Este processo é repetido até que a condição dada torna-se falso.

**fluxograma:**



### Exemplos

var index = 15

repeat{

print( "index contem o valor \(index)")

index = index + 1

}while index < 20

A saída de execução de programa acima é:

index contem o valor 15

index contem o valor 16

index contem o valor 17

index contem o valor 18

index contem o valor 19

# Instrução Continue

A instrução continue tras uma iteração do loop e para imediatamente este ciclo fazendo começar de novo a próxima iteração.

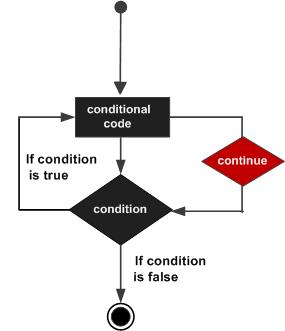
Para **o loop for, continue declaração incremento após a execução da declaração ainda será executado.** Para **While e do ... while-loop, vai continuar a re-executar a instrução condicional comunicado.**

### gramática

Sintaxe da instrução é a seguinte:

continue

**fluxograma:**



### Exemplos

var index = 10

repeat{

index = index + 1

if( index == 15 ){ // index é igual a 15 e não exibe

continue

}

print( "index contem o valor \(index)")

}while index < 20

A saída de execução de programa acima é:

index contem o valor 11

index contem o valor 12

index contem o valor 13

index contem o valor 14

index contem o valor 16

index contem o valor 17

index contem o valor 18

index contem o valor 19

index contem o valor 20

# Break

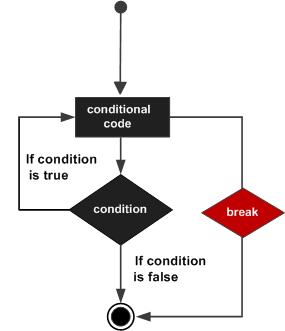
A instrução break Swift irá imediatamente finalizar a execução de todo o fluxo de controle.

### gramática

sintaxe instrução break Swift é a seguinte:

break

**fluxograma:**



### Exemplos

var index = 10

repeat{

index = index + 1

if( index == 15 ){ // index é igual a 15 para o fluxo

break

}

print( "index contem o valor \(index)")

}while index < 20

A saída de execução de programa acima é:

index contem o valor 11

index contem o valor 12

index contem o valor 13

index contem o valor 14

# Funções

As funções em Swift são usadas para realizar tarefas específicas em blocos de código separados.

Swift usa uma sintaxe unificada para expressar um simples funções de estilo C a complexa abordagem de estilo de linguagem Objective-C.

* declaração de função: dizer o nome da função compilador, tipo de retorno e os parâmetros.
* definição de função: Fornecer as funções de entidade.

## **definição de função**

funções são definidas usando a palavra-chave **func.**

Voce pode especificar um ou mais parâmetros de entrada e um tipo de valor de retorno.

Cada função tem um nome para descreve-la, o valor do parâmetro do nome da função e do tipo de chamada.

Ordem dos argumentos passados na lista de parâmetros da função deve ser a mesma**-> Depois de definir o tipo de retorno da função.**

### gramática

func funcname(parametros) -> tipoRetorno

{

Statement1

Statement2

……

Statement N

return parametro

}

### Exemplos

Abaixo defini uma função chamada função teste, os parâmetros do tipo de dados String, o valor de retorno de string:

func teste(site: String) -> String {

return site

}

print(teste("www.w3big.com"))

A saída de execução de programa acima é:

www.w3big.com

## **chamada de função**

Podemos chamar a função através do nome da função e parâmetro correspondente ao tipo da lista de parâmetros. Ordem de parâmetro de deve ser o mesmo.

# Argumentos da função

Função pode aceitar um ou mais parâmetros, também podemos usar uma tupla (tupla) passar um ou mais parâmetros para funções:

func mult(no1: Int, no2: Int) -> Int {

return no1\*no2

}

print(mult(2, no2:20))

print(mult(3, no2:15))

print(mult(4, no2:30))

A saída de execução de programa acima é:

40

45

120

# Função sem parâmetros

Nós podemos criar uma função sem parâmetros.

### sintaxe:

func nomeFuncao() -> tipoDado {

return tipoDado

}

### Exemplos

func sitename() -> String {

return "www.etecembu.com.br"

}

print(sitename())

A saída de execução de programa acima é:

www.etecembu.com.br

## **Tupla para retornar o valor que a função**

Função com tipo de valor de retorno pode ser uma string, integer, o tipo de ponto flutuante.

Tupla é semelhante à matriz, a diferença é, os elementos de tupla pode ser de qualquer tipo, utilizando parênteses.

Você pode ter vários tipos de valores como um valor composto retornado da tupla função (tupla).

O exemplo abaixo define de uma chamada MinMax (\_ :) função do papel é o de identificar os valores mínimo e máximo em uma matriz Int.

func minMax(array: [Int]) -> (min: Int, max: Int) {

var atualMin = array[0]

var atualMax = array[0]

for valor in array[1..<array.count] {

if valor < atualMin {

atualMin = value

} else if valor > atualMax {

atualMax = valor

}

}

return (atualMin, atualMax)

}

let result = minMax([8, -6, 2, 109, 3, 71])

print("menor valor \( result.min) ，maior valor \( result.max)")

MinMax (\_ :) função retorna uma tupla de dois valores Int, você pode acessá-los pelo nome desses valores, eles são marcados como mínimo e máximo, para que a consulta devolve o valor da função no.

A saída de execução de programa acima é:

menor valor -6 ，maior valor 109

## **Nenhum valor de retorno da função**

Aqui está teste (\_ :) uma outra versão da função, a função recebe o site dos parâmetros, tipo de retorno não é especificado, e saída direta valor de String em vez de devolvê-lo:

func teste(site: String) {

print("O site é：\(site)")

}

teste("http://www.w3big.com")

A saída de execução de programa acima é:

O site é：http://www.w3big.com

## **Função Nome do parâmetro**

argumento de função tem um nome de parâmetro externo e um nome de parâmetro local.

### Nome do parâmetro local

nome do parâmetro local na função da aplicação interna de usar.

func sample(number: Int) {

println(number)

}

O número do exemplo acima de nome de parâmetro local só pode ser usado no corpo da função.

func simples(numero: Int) {

print(numero)

}

simples (1)

simples (2)

simples (3)

A saída de execução de programa acima é:

1

2

3

### nome do parâmetro externo

Você pode especificar um nome de parâmetro externo na frente dos nomes de parâmetros locais, separados por um espaço, os nomes dos parâmetros externos para os parâmetros passados ​​para a função quando a função é chamada.

Você pode definir o seguinte, dois nome do parâmetro e chamá-lo:

func teste(primeiroArg a: Int, segundoArg b: Int) -> Int {

var res = a

for \_ in 1..<b {

res = res \* a

}

print(res)

return res

}

teste(primeiroArg:5, segundoArg:3)

A saída de execução de programa acima é:

125

Note que se você fornecer um nome de parâmetro externo, em seguida, a função é chamada, você deve usar um nome de parâmetro externo.

# funções aninhadas

função aninhada refere-se à definição de uma nova função dentro da função, função externa pode chamar funções dentro de uma definição de função.

Exemplos são os seguintes:

func calcDecremento(forDecremento total: Int) -> () -> Int {

var aposDecremento = 0

func decremento() -> Int {

aposDecremento -= total

return aposDecremento

}

return decremento

}

let decrem = calcDecremento(forDecrement: 30)

print(decrem())

A saída de execução de programa acima é:

-30

# Iniciando o estudo de OO em Swift

Com as explicações sobre cada um dos pontos vistas em aula, representarei aqui uma breve recordação sobre os pontos e trarei exemplos de como aplicar.

# Classe

Classes são os códigos usado para construir uma estrutura geral e flexível.

Nesta estrutura de classes podemos definir as propriedades (constantes, variáveis) e métodos.

E outras linguagens de programação são diferentes, Swift não exige a criação de classes personalizadas para arquivos de interface e implementação separados. Você tem apenas que definir uma classe em um único arquivo, assim o sistema irá gerar automaticamente o código para outras interfaces externas.

## **tipo de corpo e comparação da estrutura**

Classes têm muito em comum com as estruturas que vimos ate o momento:

* Atributos são definidos e utilizados para armazenar valores
* Define um método para proporcionar funcionalidade
* Script definido é usado para acessar o valor da subsidiária
* Definir o construtor é usado para gerar valor de inicialização
* Podemos estender as funções implementadas a fim de aumentar o padrão.

Comparado com a estrutura, classe tem as seguintes características adicionais:

* Herança permite que uma classe herda de outra característica de classe
* Conversão de tipo que permite em tempo de tempo de execução a verificação e interpretação de uma instância de classe
* Contagem de referência, permite que várias referências (objetos) sejam criados a partir de uma classe

### sintaxe:

Class NomeClasse {

Definicao 1

Definicao 2

……

Definicao N

}

### definição de classe

class Estudante{

var nomeEstudante: String

var idade: Int

var Altura: Double

}

Instanciar a classe:

let criarEstudante = Estudante()

### Exemplos

class Estudante {

var nome: String

init(nome: String) {

self.nome = nome

}

}

let estudante1 = Estudante(nome: “Fulano”)

print("O nome do estudante é \( estudante1.nome)")

A saída de execução de programa acima é:

O nome do estudante é Fulano

# Métodos

# Métodos da instância

Em linguagem Swift, um método de instância pertence a uma classe particular, uma estrutura ou instâncias tipo enum.

método de instância fornece os seguintes métodos:

* Você pode acessar e modificar propriedades de instância
* Exemplo fornecendo funções relacionadas ao objeto

Exemplos de métodos para escrever em torno dele, deve estar entre os tipos de chaves ({}).

Exemplos de método implícito pode acessar todos os métodos e propriedades de outras instâncias do tipo a que pertence.

Exemplos de métodos só pode ser chamado de uma instância específica da classe a que pertence.

método de instância não pode ser separada na instância existente é chamado.

### gramática

func nomeMetodo(Parametros) -> tipoRetorno

{

Statement1

Statement2

……

Statement N

return parametro

}

### Exemplos

class Counter {

var count = 0

func incremento() {

count++

}

func incrementoPor(valor: Int) {

count += valor

}

func reset() {

count = 0

}

}

let counter = Counter() //Cria uma instancia da classe

counter.incremento() //O objeto chama a função “incremento”

counter.incrementoPor(5) //Chama a função “incrementoPor”

print(counter.count) //Imprimi o valor armezenado do retorno da funçao

counter.reset()// Zera o valor do contador a partir da funcao

print(counter.count)// Imprimi o valor do contador

A saída de execução de programa acima é:

6

0

classe Contador define três métodos de instância:

* incremento de modo que o contador é incrementado por um;
* incrementoPor(valor: Int) fazem contador por um valor inteiro especificado é incrementado;
* reset o contador é reposto a 0.

Counter Esta classe também declara um atributo variável count , usá-lo para controlar o valor atual do contador.

# Construtores / Inicializadores

O processo de construção é usar uma instância de uma classe, realizando o processo de preparação. Este processo envolve a definição de valores iniciais para cada atributo e a instância para a preparação e implementação das tarefas de inicialização necessários.

## **A atribuição inicial da propriedade de tipo de armazenamento**

Quando uma instância é criada, ela deve definir o valor inicial adequado para todos os atributos do tipo de armazenamento.

Ao armazenar, a atribuição de propriedades no construtor e os seus valores são definidos diretamente e não desencadeiam qualquer observador propriedade.

processo de atribuição de propriedade de armazenamento no construtor:

* Criando o valor inicial.
* Na definição de atributo especifica os valores de propriedade padrão.
* Inicializa uma instância, e chama o método init ().

## **construtor**

O construtor é chamado quando você cria uma nova instância de um tipo particular. Sua forma mais simples é semelhante a um método de instância sem quaisquer parâmetros, o nome de **init** como palavra-chave.

### gramática

init()

{

// comandos

}

### Exemplos

A estrutura a seguir define init como um construtor sem argumentos, e no interior do método é definido o comprimento do atributo e a largura de inicialização com os valores de 6 e 12.

Podemos definir os valores iniciais para a propriedade do tipo de armazenamento no construtor; Além disso, é possível definir um valor padrão quando uma declaração de propriedade.

Ele permite que você use o construtor padrão, mais claro, e pode automaticamente inferir pelo atributo tipo de valor predefinido.

O exemplo a seguir queremos definir o valor padrão quando uma declaração de propriedade:

class retangulo {

var altura: Double

var largura: Double

init() {

altura = 6

largura= 12

}

}

var area = retangulo ()

print("A área do retângulo é \(area.length\*area.breadth)")

A saída de execução de programa acima é:

A área do retângulo é 72.0

## **Parâmetros de configuração**

Você pode definir o construtor desde que passa os parâmetros do construtor init (), como segue:

class Retangulo {

var altura: Double

var largura: Double

var area: Double

init(altura: Double, largura: Double) {

self. altura = altura

self. largura = largura

area = altura \* largura

}

}

let ar = Rectangulo(altura: 6, largura: 12)

print(“A área do retângulo é: \(ar.area)")

A saída de execução de programa acima é:

A área do retângulo é: 72.0

# Processo de herança e construção da classe

O Swift fornece dois tipos de construtores de classe para assegurar que todas as instâncias de classe e as propriedades de armazenamento possam ser obtidas nos valores iniciais, que são especificados construtor e construtor de conveniência.

|  |  |
| --- | --- |
| Especifica o construtor | construtor conveniência |
| O construtor da classe principal | Classe relativamente menor, construtor assistida |
| Toda a propriedade é inicializada classe ofereceu-se e chamar o construtor da classe pai com base na cadeia de pais para conseguir a inicialização da classe pai. | Você pode definir construtor conveniência de chamar a mesma classe, conforme especificado no construtor, e fornece valores padrão para parâmetros. Você também pode definir construtor conveniência para criar uma instância de um propósito particular ou insumos específicos. |
| Cada classe deve ter pelo menos um construtor especificado | Somente quando necessário para facilitar o construtor da classe |
| Init(parameters) { | convenience init(parameters) { |
| statements | statements |
| } | } |

### 

### Especifica o construtor de instância

class mainClass {

var n1 : Int

init(n1 : Int) {

self.n1 = n1

}

}

class subClass : mainClass {

var n2 : Int

init(n1 : Int, n2 : Int) {

self.n2 = n2

super.init(n1:n1)

}

}

let res = mainClass(n1: 10)

let res2 = subClass(n1: 10, n2: 20)

print("res vale: \(res.no1)")

print("res2 vale: \(res2.no1)")

print("res2 vale: \(res2.no2)")

A saída de execução de programa acima é:

res 为: 10

res 为: 10

res 为: 20

# Herança Swift

A herança pode ser entendida como uma classe que pode obter os métodos e propriedades de outra classe.

Quando uma classe herda de outra classe, a classe derivada é chamada de uma subclasse, a classe hereditária chamada superclasse (ou pai)

Em Swift, as classes podem ter acesso e chamar os métodos e propriedades da superclasse, e podem substituí-los.

Nós também podemos adicionar propriedades além das herdadas para a classe.

# classe base

Não há outras classes herdar uma classe, chamar a classe base (Base Class).

O exemplo a seguir, definimos uma classe “Estudante” como classe base, descreve o estudante (stname) e todas as suas notas (n1, n2, n3):

class Estudante {

var stname: String

var n1: Int

var n2: Int

var n3: Int

init(stname: String, n1: Int, n2: Int, n3: Int) {

self.stname = stname

self.n1 = n1

self.n2 = n2

self.n3 = n3

}

}

let stname = "swift"

let n1 = 9

let n2 = 8

let n3 = 6

print(stname)

# Subclasse

Refere-se a uma subclasse quando é criar uma nova classe com base em uma classe existente.

Para especificar a superclasse de uma classe, o nome de classe Super escrito na parte de trás do nome da subclasse, (:), separados por dois pontos na sintaxe, como segue:

class Subclasse: Superclass {

comandos

}

### Exemplos

O exemplo a seguir, definimos uma superclasse Estudante e então uma subclasse Tom que ira herdá-la:

class Estudante {

var nome: String

var n1: Int

var n2: Int

init(nome: String, n1: Int, n2: Int) {

self.nome = nome

self.n1 = n1

self.n2 = n2

}

func mostrar()

{

print("n1:\(self.n1), n2:\(self.n2)")

}

}

class Tom : Estudante

{

init()

{

super.init(nome: “Fulano”, n1: 10, n2: 7)

}

}

let tom = Tom()

tom.show()

A saída de execução de programa acima é:

n1:10, n2:7

# Rewrite (Overriding)

Como vimos em uma subclasse podem ser herdados métodos de instância, métodos de classe, instância e as propriedade, mas para realizar as suas capacidades de personalização, chamamos este comportamento reescrito (primordial).

Podemos usar a palavra-chave **override** para alcançar reescrita.

# Substituindo métodos e propriedades

### Substituindo métodos

Em nossa sub-classe, podemos usar a palavra-chave **override** para substituir o método da superclasse.

Os exemplos a seguir, reescrevemos o método show ():

class SuperClass {

func show() {

print("método da SuperClass")

}

}

class SubClass: SuperClass {

override func show() {

print("Metodo da SubClass")

}

}

let superClass = SuperClass()

superClass.show()

let subClass = SubClass()

subClass.show()

A saída de execução de programa acima é:

Método da SuperClass

Método da SubClass