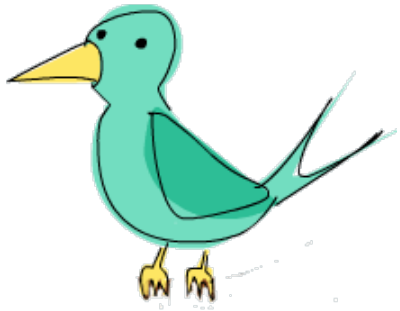
Índice Começando

# Introdução

## Sobre este tutorial

Bem-vindo ao **Aprender Haskell será um grande bem para você**! Se você já esta lendo isto, existem boas chances de você querer aprender Haskell. Muito bem, você já esta no lugar certo, mas antes vamos conversar um pouco sobre este tutorial.

Eu decidi escrever porque estava querendo solidificar meus conhecimentos em Haskell e porque pensei que poderia ajudar as pessoas novas em Haskell a aprende-lo sob a minha perspectiva. Este é só mais um tutorial sobre Haskell pairando pela internet. Quando eu comecei em Haskell eu não aprendi a partir de apenas uma fonte. O meu caminho para o aprendizado passou pela leitura de diversos e diferentes tutoriais e artigos porque cada um explicava algo seguindo por um caminho diferente do outro. Ao passar por diferentes fontes, eu consegui juntar as peças e tudo acabou caindo no mesmo lugar. Portanto, esta é apenas mais uma fonte adicional a se utilizar para aprender Haskell para que você tenha uma maior chance de encontrar uma que você realmente goste.



Este tutorial é destinado a pessoas que tenham experiência em linguagens de programação imperativa (C, C++, Java, Python …) sem terem programado antes em uma linguagem funcional (Haskell, ML, OCaml …). Apesar de que eu aposto que se você não tiver uma larga experiência em programação, um rápido capítulo como o que você irá acompanhar vai habilitá-lo a aprender Haskell.

O canal #haskell (ou #haskell-br) na rede freenode é um belo local para mandar alguma questão caso você estiver emperrado. As pessoas lá são extremamente legais, pacienciosas e entendem os newbies.

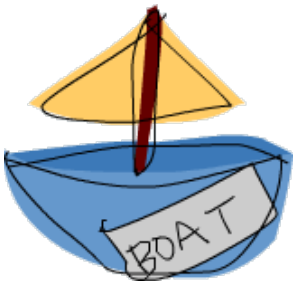
Eu falhei aproximadamente 2 vezes antes de finalmente aprender Haskell, isto porque tudo me parecia muito estranho e eu não conseguia entender. Mas logo em seguida, após ultrapassar essa primeira barreira, me veio o "estalo" e então entendi que era tudo muito fácil. Acho que o que estou tentando dizer é o seguinte: Haskell é muito legal e se você realmente tiver interesse em programação você deve continuar mesmo que ele lhe pareça estranho. Aprender Haskell é muito parecido com quando se está aprendendo a programar pela primeira vez — isto que é divertido! Ele te força a pensar diferente, o que nos remete ao próximo capítulo …

## Então, o que é Haskell?


Haskell é uma **linguagem de programação puramente funcional**. Em linguagens de programação imperativas você pensa nas coisas seguindo uma sequência computacional de tarefas sendo executadas, embora durante o processo possam mudar de estado. Por exemplo, você define uma variável como 5 e então faz alguma coisa e em seguida a define como sendo alguma outra coisa qualquer. Você possui o controle do fluxo da estrutura podendo fazer uma determinada ação diversas vezes. Na programação puramente funcional

você não diz para o computador o que fazer, porém muitas vezes você diz em qual coisa está. O fatorial de um número é o produto de todos os números sobre 1 deste número, a soma de uma lista de números é o primeiro mais a soma de todos os outros números e assim por diante. Você expressa isto na forma de funções. Você também não pode definir uma variável como sendo alguma coisa e em seguida defini-lá como sendo alguma outra coisa mais tarde. Se você disser a uma função que algo é 5, você não poderá dizer depois que é alguma outra coisa porque você já disse que era 5. O que você é? Algum tipo de mentiroso? Então, em linguagens puramente funcionais, uma função não tem efeitos colaterais. A única coisa que podemos fazer com uma função é calcular algo e devolvê-lo como um resultado. Inicialmente, eu vi isto como uma limitação, porém isto realmente tem algumas consequências interessantes: se a função é chamada duas vezes com os mesmos parâmetros, isto garantirá o retorno de um mesmo resultado. Isso se chama **transparência referencial** e não só permite que o compilador raciocine sobre o comportamento do programa, como também permite que você deduza facilmente (e até mesmo prove) que uma função está correta e, em seguida, construa funções mais complexas juntando diversas funções simples por "colagem.

Haskell é **preguiçoso**. Isso significa que a menos que seja especificamente dito de outra forma, Haskell não irá executar funções e calcular as coisas antes que ele seja realmente obrigado a lhe mostrar um resultado. Isto vai bem de encontro com a transparência referencial que lhe permite pensar em programas como uma série de **transformações nos dados**. Isto também permite simplificar coisas como estruturas de dados infinitas. Digamos que você tem uma lista de números imutáveis `xs = [1,2,3,4,5,6,7,8]` e uma função `doubleMe` que multiplica cada elemento por 2 e em seguida retorna uma nova lista. Caso quiséssemos multiplicar nossa lista por 8 em uma linguagem imperativa fazendo `doubleMe (doubleMe (doubleMe (xs )))`, ele provavelmente iria passar uma vez pela lista, fazer uma cópia e retornar. Em seguida, ele iria passar por mais duas vezes e retornar o resultado. Em uma linguagem preguiçosa, chamando `doubleMe` numa lista sem forçar para que seja mostrado algum resultado assim que acabar, ela irá lhe dizer "Sim sim, faço isso mais tarde". Mas uma vez que você queira ver o resultado, o primeiro `doubleMe` dirá para o segundo que quer o resultado do 1, agora! O segundo 1 irá dizer para o terceiro 1 e o terceiro 1 relutantemente irá retornar o 1 duplicado, com um 2. O segundo 1 recebido irá retornar um 4 para o primeiro. O primeiro 1 olhará aquilo e dirá para você que o primeiro elemento é 8. Por isso, ele só passa uma vez pela lista e só quando você realmente precisa dela. Dessa maneira, quando você quiser alguma coisa em uma linguagem "preguiçosa", você poderá ter apenas alguns dados iniciais e eficientemente transformá-los e melhorá-los para que eles se assemelhem com aquilo que você quer no final.




Haskell é **estaticamente tipado**. Quando você compilar seu programa, o compilador saberá quais partes do código é um número, o que é uma string e assim por diante. Isso significa que uma série de possíveis erros poderão ser capturados em tempo de compilação. Se você tentar adicionar um número a uma string, o compilador irá se queixar de você. Haskell usa um bom sistema de tipos que tem **inferência de tipo**. Isso significa que você não precisa explicitamente identificar cada pedaço de código com um tipo porque o sistema de tipos inteligente descobrirá muito sobre ele. Se você disser `a = 5 + 4` , você não precisará dizer para o Haskell que `a` é um número, ele irá descobrir isso por si só. Inferência de tipo também permite que o seu código seja mais genérico. Se você fizer uma função de soma com dois parâmetros e não declarar explicitamente seus tipos, ela irá funcionar com quaisquer valores que sejam números.

Haskell é **elegante e conciso**. Isto porque ele utiliza uma série de conceitos de alto nível, programas Haskell são normalmente mais curtos do que os seus equivalentes imperativos. E programas mais curtos são mais fáceis de se

manter e têm menos bugs.

Haskell foi feito por **caras realmente inteligentes** (com doutorado). O trabalho sobre Haskell começou em 1987 quando uma comissão de pesquisadores se reuniu para projetar uma linguagem matadora. Em 2003, o Relatório Haskell foi publicado já com uma versão estável da linguagem.

## O que eu preciso para embarcar nessa

Um editor de texto e um compilador Haskell. Provavelmente você já tem um editor de texto favorito instalado, então não perca tempo com isso. Neste tutorial iremos usar o GHC, o compilador Haskell mais usado. O melhor modo de iniciar na linguagem é baixando o Haskell Platform, que é algo como o Haskell com esteróides.

GHC pode pegar um script Haskell (que normalmente tem uma extensão .hs) e compilá-lo, mas ainda existe um modo interativo que permite que você interativamente interaja com os scripts. Interativamente. Você pode chamar funções de scripts carregados que os resultados serão exibidos imediatamente. Para o seu aprendizado é muito mais fácil e rápido que compilar toda vez que fizer uma alteração e enfim executar o programa a partir do prompt. O modo interativo é chamado digitando `ghci` no seu prompt. Se você definir algumas funções em um arquivo chamado, digamos, `myfunctions.hs`, você poderá carregar essas funções digitando `: l myfunctions` para então poder brincar com elas, desde que `myfunctions.hs` esteja na mesma pasta na qual o `ghci` foi invocado. Se você mudar o seu script ".hs, basta executar `: l myfunctions` novamente ou fazer `: r` , que é o equivalente a recarregar o script atual. O processo comum para mim quando estou brincando, é definir algumas funções em um arquivo .hs, carregá-lo e modificá-lo sem pudor. Quando preciso, carrego um outro arquivo .hs novamente e assim por diante. Isto é também o que faremos aqui.

Índice Começando