

# Introdução

## BCC22 Programação Funcional — Aula Prática 00

Lucília Camarão de Figueiredo

A prática deve ser realizada antes da aula prevista para a discussão da mesma. Verifique a data prevista para a discussão de cada aula prática no calendário do curso disponível em [www.dcc.ufmg.br/~lucilia/cursos/func](http://www.dcc.ufmg.br/~lucilia/cursos/func). Traga para a aula de discussão o código que você implementou. A discussão não será eficaz se você não tentar fazer os exercícios previamente.

Você pode trabalhar em conjunto com outros colegas, mas deve entender o que está fazendo, pois não irá contar com a ajuda de colegas durante os exames.

A realização das práticas e presença nas aulas são obrigatórias e valem pontos na disciplina.

## 1 Bem-vindo

Bem-vindo ao seu primeiro exercício de programação funcional! Esse exercício tem como objetivo orientá-lo sobre como escrever os seus primeiros programas em Haskell e executá-los no interpretador **GHCI**.

O texto desta prática consiste de quatro partes:

1. **GHCI** Informações sobre as ferramentas básica de programação a serem usadas no curso.
2. **Iniciando** Um exercício simples, usando funções aritméticas em Haskell.
3. **QuickCheck** Uma biblioteca para teste automático de programas
4. **Submetendo o seu trabalho** Instruções sobre submissão de trabalhos das práticas.

## 2 GHCI

**GHCI**— Glasgow Haskell Compiler — é uma implementação da linguagem de programação **Haskell**. O **GHCI** é interativo: ele avalia cada expressão que você digita no *prompt* do interpretador e exibe o resultado. Você pode iniciar a execução do **GHCI** digitando `ghci` no prompt de comandos do sistema, ou, alternativamente, a partir do menu de programas.

### Exercícios

1. Abra uma janela de terminal de comando e digite “ghci” ou selecione o programa a partir do menu de programas. Ao iniciar, o GHCI irá exibir a seguinte mensagem:

```
GHCI, version 6.10.4: http://www.haskell.org/ghc/ :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer ... linking ... done.
Loading package base ... linking ... done.
Prelude>
```

Esse é o ambiente interativo do GHCI. Primeiramente, alguns exercícios simples.

- (a) Digite “ $3 + 4$ ” no prompt. Qual o resultado?

- (b) Tente “ $3 + 4 * 5$ ” e “ $(3 + 4) * 5$ ”.

A avaliação de expressões aritméticas em Haskell funciona como esperado?

O ambiente interativo pode avaliar qualquer expressão Haskell, não apenas expressões aritméticas:

```
Prelude> length "This is a string"
16
Prelude> reverse "This is not a palindrome"
"emordnilap a ton si sihT"
```

Além de expressões Haskell, o GHCi entende uma variedade de comandos, por exemplo:

<code>:load <i>filename</i></code>	Load a file containing Haskell definitions
<code>:reload</code>	Reload the most recently-loaded file
<code>:type <i>expression</i></code>	Display the type of expression
<code>:?</code>	Display a list of commands

### Exercício

2. Encontre qual é o comando para terminar o GHCi e use esse comando para terminá-lo.

### Editor de Programas

Para editar seus programas, você pode usar o editor de textos de sua preferência. Entretanto, recomendamos usar um editor que possa ser configurado para edição de programas Haskell, tal como **Notepad++**, no Windows, ou **Emacs**, no Linux.

## 3 Iniciando

Faça o download do arquivo `pratica00.zip` do site da disciplina: <http://www.dcc.ufmg.br/~lucilia/cursos/func>. Esse arquivo deve conter este documento (`pratica00.pdf`) e o arquivo `pratica00.hs`. Abra o arquivo `pratica00.hs` no editor de textos. Abaixo dos comentários introdutórios e da frase `import Test.QuickCheck`, que carrega a biblioteca `QuickCheck` a ser usada posteriormente, você verá a seguinte definição de função:

```
double :: Int -> Int
double x = x + x
```

### Exercício

3. (a) Parte da definição (a linha `double x = x + x`) está indentada incorretamente: a definição da função deveria estar alinhada verticalmente com a sua assinatura de tipo (a linha acima). Corrija esse erro e salve o programa.
- (b) Carregue o arquivo `pratica00.hs` no GHCi. Use o GHCi para obter:
- o dobro de 21
  - o tipo da função `double`
  - o tipo da expressão `double 21`
- (c) O que acontece se você pede para avaliar a expressão `double "three"`?
- (d) Complete a definição de `square :: Int -> Int` no arquivo `pratica00.hs` de modo que essa função compute o quadrado de um número (você deve substituir a palavra “undefined”). Recarregue o seu programa e teste a sua definição.

## Triplas Pitagóricas

Pitágoras foi um matemático Grego que viveu por volta de 570 a 490 AC. Ele é conhecido pelos estudantes de primeiro grau como o descobridor da relação entre os lados de um triângulo retângulo. Existe entretanto pouca evidência de que Pitágoras tenha sido de fato um

geômetra. As referências mais remotas a Pitágoras não fazem qualquer menção à sua capacidade matemática, mas sim às suas recomendações dietárias, tal como proibir seus seguidores de comer feijão.

Tenha ou não descoberto o teorema que leva seu nome, ele certamente tornou-se conhecido na antiguidade. Uma tábua de pedra da Mesopotâmia, pelo menos 1000 anos anterior a Pitágoras, contém parte da lista das chamadas “triplas Pitagóricas”: inteiros positivos que correspondem a lados de um triângulo retângulo.

Voltando aos gregos, Euclides (325–265 BC) descreveu um método para gerar as triplas Pitagóricas no seu famoso tratado “Os Elementos”. Nessa parte do exercício você vai adotar uma abordagem mais moderna para o velho problema de gerar e verificar triplas Pitagóricas, usando Haskell. Primeiramente, uma definição formal: uma tripla Pitagórica é um conjunto de três inteiros positivos  $(a, b, c)$  que satisfazem a equação  $a^2 + b^2 = c^2$ . Por exemplo,  $(3, 4, 5)$  é uma tripla Pitagórica, já que  $3^2 + 4^2 = 9 + 16 = 25 = 5^2$ .

### Exercício

4. Defina a função `isTriple` que implementa o teste para triplas Pitagóricas. Você não precisa considerar o caso de triplas com valores negativos ou zero.

- (a) No arquivo `pratica00.hs`, encontre a assinatura de tipo `isTriple :: Int -> Int -> Int -> Bool` e substitua “undefined” pela definição apropriada (use `==` para testar se dois valores são iguais).
- (b) Carregue seu programa no GHCi. Teste sua função para alguns valores de entrada apropriados. Certifique-se de que ela retorna `True` para números que satisfazem a equação (tais como 3, 4 e 5), e retorna `False` para números que não satisfazem a equação (tais como 3, 4 e 6).

```
Main> isTriple 3 4 5
True
Main> isTriple 3 4 6
False
```

Agora vamos gerar algumas triplas automaticamente. Uma maneira simples de fazer isso é a seguinte:  $(x^2 - y^2, 2yx, x^2 + y^2)$  é uma tripla Pitagórica para todos os inteiros positivos  $x$  e  $y$  onde  $x > y$ . O requerimento de que  $x$  e  $y$  sejam positivos e que  $x > y$  garante que os lados do triângulo são positivos; neste exercício, vamos por enquanto desconsiderar esse teste.

### Exercício

5. Escreva funções `leg1`, `leg2` e `hyp` que geram os componentes da tripla Pitagórica usando as fórmulas acima.

- (a) Usando as fórmulas, adicione ao seu arquivo `pratica00.hs` definições adequadas de

```
leg1 :: Int -> Int -> Int
leg2 :: Int -> Int -> Int
hyp  :: Int -> Int -> Int
```

- (b) Teste suas funções para valores de entrada apropriados. Verifique se as triplas geradas são válidas.

```
Main> leg1 5 4
9
Main> leg2 5 4
40
Main> hyp 5 4
41
Main> isTriple 9 40 41
True
```

## 4 QuickCheck

Vamos agora usar a biblioteca **QuickCheck** para testar se a sua combinação de `leg1`, `leg2` e `hyp` de fato cria uma tripla Pitagórica. QuickCheck pode testar sua função para uma grande quantidade de dados, gerados aleatoriamente. Antes de começar a usar o QuickCheck, vamos experimentar alguns testes da sua função manualmente.

### Exercício

6. A função `prop_triple` — o nome começa com `prop`(riedade) para indicar que é para ser usada pelo QuickCheck — usa as funções `leg1`, `leg2`, `hyp` para gerar uma tripla Pitagórica, e usa a função `isTriple` para verificar se o valor gerado é de fato uma tripla Pitagórica.
  - (a) Como essa função funciona? Que tipo de entrada ela espera e que tipo de saída ela gera?
  - (b) Teste essa função para pelo menos 3 entradas apropriadas. Que resultados você espera para cada uma?
  - (c) Digite o seguinte no prompt do GHCi (o ‘C’ deve ser maiúsculo):

```
Main> quickCheck prop_triple
```

O comando anterior faz com que QuickCheck execute cem testes aleatórios para a sua função teste. Se o resultado for:

```
OK, passed 100 tests.
```

então tudo está ok. Caso contrário QuickCheck emite uma resposta tal como:

```
Falsifiable, after 0 tests:
5
6
```

Isso significa que a sua função falhou quando QuickCheck tentou avaliá-la com os argumentos 5 e 6. Se o mesmo teste fosse feito manualmente, o resultado seria:

```
Main> prop_triple 5 6
False
```

Se isso acontecer, pelo menos uma das funções anteriores, `isTriple`, `leg1`, `leg2` e `hyp` tem algum erro, que você deverá descobrir e corrigir.

## 5 Submissão do trabalho

Quando você terminar seu exercício, escreva o seu nome e número de matrícula no lugar indicado, no início do arquivo `pratica00.hs`. A submissão de cada prática deve ser feita sempre até o dia anterior ao marcado para a discussão da mesma, conforme o programa de aulas disponível no site da disciplina <http://www.dcc.ufmg.br/~lucilia/cursos/func>. Se você não tiver conseguido terminar, submeta seu trabalho mesmo assim, ainda que incompleto, pois não serão aceitos trabalhos submetidos fora do prazo. Entretanto, antes de submeter seu trabalho certifique-se que ele compila corretamente no GHCi. Se parte do código não compila corretamente, você pode comentá-lo, usando `--` no início da linha.

— Incluir instruções de submissão