

Definição de funções simples

1.1 Para que três valores possam ser medidas dos lados de um triângulo deve verificar-se a seguinte condição: qualquer dos valores deve ser inferior à soma dos outros dois. Complete a definição de uma função que testa esta condição; o resultado deve ser um valor booleano (**True** ou **False**).

```
testaTriangulo :: Float -> Float -> Float -> Bool
testaTriangulo a b c = ...
```

1.2 Podemos calcular a área A de um triângulo de lados a , b , c usando a fórmula de Heron:

$$A = \sqrt{s(s-a)(s-b)(s-c)},$$

onde $s = (a + b + c)/2$. Complete a seguinte definição em Haskell duma função para calcular esta área.

```
areaTriangulo :: Float -> Float -> Float -> Float
areaTriangulo a b c = ...
```

1.3 Usando as funções **length**, **take**, **drop** apresentadas na primeira aula, escreva uma função **metades** que divide uma lista em duas com metade do comprimento (aproximadamente). Exemplo:

```
metades [1,2,3,4,5,6,7,8] == ([1,2,3,4], [5,6,7,8])
```

Experimente a sua definição no interpretador e investigue o acontece se a lista tiver comprimento ímpar.

1.4 Neste exercício pretende-se que use as funções o prelúdio-padrão de processamento de lista apresentadas na primeira aula: **head**, **tail**, **length**, **take**, **drop** e **reverse**.

- (a) Mostre que a função **last** (que obtém o último elemento de uma lista) pode ser escrita como composição de algumas das funções acima. Consegue encontrar *duas* definições diferentes?
- (b) Analogamente, mostre que a função **init** (que remove o último elemento duma lista) pode ser definida usando as funções acima de duas formas diferentes.

1.5 Os coeficientes binomiais $\binom{n}{k}$ são os números que aparecem como coeficientes dos termos X^k na expansão de $(1 + X)^n$; correspondem também ao *número de formas distintas de escolher k objetos entre n* (ou, equivalentemente, o número de subconjuntos com k elementos que podemos formar de um conjunto de n elementos). Neste exercício pretende-se calcular estes coeficientes para quaisquer n e k .¹

¹Usamos inteiros de precisão arbitrária (**Integer**) para evitar “overflow” nos produtos.

- (a) Complete a definição duma função

```
binom :: Integer -> Integer -> Integer
binom n k = ...
```

para calcular o coeficientes binomial de n e k pela seguinte fórmula:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Sugestão: pode exprimir $n!$ como `product [1..n]`.

- (b) Podemos escrever a fórmula acima para calcular o mesmo resultado mas evitando multiplicações desnecessárias. Por exemplo, podemos calcular $\binom{10}{2}$ sem ter de calcular $10!$ e $8!$:

$$\binom{10}{2} = \frac{10!}{2! \times 8!} = \frac{10 \times 9 \times 8!}{2 \times 1 \times 8!} = \frac{10 \times 9}{2} = 45$$

Usando esta simplificação escreva uma definição alternativa `binom'` com o mesmo tipo e que produz os mesmos resultados mas efetuando menos cálculos.

Sugestão: Se $k < n - k$, o numerador reduz-se a ao produto dos números de $n - k + 1$ até n e o denominador a $k!$. No caso em que $k \geq n - k$, o numerador reduz-se ao produto dos números de $k + 1$ até n e o denominador a $(n - k)!$.

1.6 Podemos calcular as raízes de uma equação de 2º grau $aX^2 + bX + c = 0$ pela fórmula resolvente:

$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Defina uma função

```
raizes :: Float -> Float -> Float -> (Float, Float)
raizes a b c = ...
```

que calcula as raízes da equação; o resultado deve ser o *par* (`x1`, `x2`) de raízes. Sugestões:

1. a função raiz quadrada em Haskell é `sqrt`;
2. pode usar `let` ou `where` para definir nomes de valores auxiliares como o binómio discriminante $b^2 - 4ac$.

Tipos e classes

1.7 Indique tipos admissíveis para os seguintes valores.

- (a) `['a', 'b', 'c']`

- (b) ('a', 'b', 'c')
- (c) [(False, '0'), (True, '1')]
- (d) [(False, True), ['0', '1']]
- (e) [tail, init, reverse]
- (f) [id, not]

1.8 Indique o tipo mais geral para as seguintes definições; tenha o cuidado de incluir restrições de classes no caso de operações com sobrecarga.

- (a) *segundo* $xs = head\ (tail\ xs)$
- (b) *trocar* $(x, y) = (y, x)$
- (c) *par* $x\ y = (x, y)$
- (d) *dobro* $x = 2 * x$
- (e) *metade* $x = x/2$
- (f) *minuscule* $x = x \geq 'a' \ \&\&\ x \leq 'z'$
- (g) *intervalo* $x\ a\ b = x \geq a \ \&\&\ x \leq b$
- (h) *palindromo* $xs = reverse\ xs == xs$
- (i) *twice* $f\ x = f\ (f\ x)$

Expressões condicionais, guardas e padrões

1.9 Escreva duas definições, respectivamente usando expressões condicionais e guardas, da função `classifica :: Int -> String` que faz corresponder uma classificação qualitativa a uma nota de 0 a 20:

≤ 9	“reprovado”
10–12	“suficiente”
13–15	“bom”
16–18	“muito bom”
19–20	“muito bom com distinção”

1.10 O *índice de massa corporal* (IMC) é uma medida simples para classificar o peso de adultos.² O IMC de um indivíduo é calculado como o valor do peso (em quilogramas) a dividir pelo quadrado da altura (em metros):

$$\text{IMC} = \text{peso} / \text{altura}^2$$

Por exemplo: um indivíduo com 70Kg e 1.70m de altura tem IMC igual a $70/1.70^2 \approx 24.22$. Classificamos o resultado nos seguinte intervalos:

²<https://www.euro.who.int/en/health-topics/disease-prevention/nutrition/a-healthy-lifestyle/body-mass-index-bmi>

	IMC	< 18.5	“baixo peso”
18.5 ≤	IMC	< 25	“peso normal”
25 ≤	IMC	< 30	“excesso de peso”
30 ≤	IMC		“obesidade”

Escreva uma definição da função `classifica :: Float -> Float -> String` que determina a classificação acima; os dois argumentos da função são, respectivamente, o peso em quilogramas e a altura em metros.

1.11 Considere duas possíveis definições das funções `max` e `min` do prelúdio-padrão que calculam, respectivamente, o máximo e o mínimo de dois valores:

```
max, min :: Ord a => a -> a -> a
max x y = if x>y then x else y
min x y = if x<y then x else y
```

- Escreva definições deste género para duas funções `max3` e `min3` para calcular, respectivamente, o máximo e o mínimo de três números.
- Observe que as operação de máximo e mínimo são *associativas*. Por exemplo, para calcular o máximo de três valores podemos determinar o máximo entre dois deles e depois o máximo do resultado com o terceiro. Re-escreva as funções `max3` `min3` usando esta ideia e as funções de `max` e `min` do prelúdio-padrão.

1.12 Escreva uma definição da função lógica ou-exclusivo

```
xor :: Bool -> Bool -> Bool
```

usando múltiplas equações com padrões.

1.13 Pretende-se implementar uma função `safetail :: [a] -> [a]` que estende a função `tail` do prelúdio de forma a dar a lista vazia quando o argumento é a lista vazia (em vez de um erro). Escreva três definições diferentes usando condicionais, equações com guardas e padrões.

1.14 Escreva duas definições da função `curta :: [a] -> Bool` para testar se uma lista tem zero, um ou dois elementos, usando:

- a função `length` do prelúdio-padrão;
- múltiplas equações e padrões.

1.15 A mediana de três valores é o valor “no meio” quando os colocamos por ordem crescente. Por exemplo: `mediana 2 3 (-1) == 2`.

- Escreva uma definição da função `mediana` para determinar a mediana de 3 valores quaisquer. Qual será o seu tipo mais geral? Note que podemos determinar a mediana usando apenas comparações de ordem.
- Em vez de definir a mediana diretamente usando comparações (que provavelmente terá sido a sua primeira ideia), pode usar o seguinte método: somamos os 3 valores e subtraímos o maior e o menor. Re-defina a função `mediana` desta forma. Qual será agora o tipo mais geral?

1.16 Defina uma função

```
converte :: Int -> String
```

para converter um inteiro positivo inferior a 1 milhão para texto em português.
Alguns exemplos:

```
converte 21 = "vinte e um"  
converte 1234 = "mil duzentos e trinta e quatro"  
converte 123456  
    = "cento e vinte e três mil quatrocentos e cinquenta e seis"
```

Ideia: Vamos começar por definir funções auxiliares para converter para texto os números inferiores a 100 e 1000.