(1) Implementar uma função que verifica se uma string contém um palindromo

```
palindromo :: String -> Bool
```

(2) A soma do comprimento de qualquer dois lados de um triângulo é sempre maior do que o comprimento do terceiro lado. Fazer uma função que recebe o comprimento dos três lados de um triângulo e verifica essa condição

```
> verificaTriangulo 2 2 2
True
>verificaTriangulo 1 4 1
False
```

(3) Defina, usando guardas, a função `sinal`

```
sinal :: Int -> Int
```

que recebe um inteiro como entrada e devolve: **-1** se a entrada for um número negativo, **1**, caso seja positivo ou **0** caso a entrada seja o número **zero**

(4) Implemente a função:

```
menorTres :: Int -> Int -> Int -> Int
```

que recebe três inteiros e devolve o menor entre os três

(5) Em Haskell, para implementarmos algum tipo de repetição, usamos recursão. Por exemplo, sabemos que o fatorial de um número `n` é calculado a partir da multiplicação desse número com todos os seus atencessores até o número 1. Por exemplo, o fatorial de 4 é calculado da seguinte forma:

```
4 * 3 * 2 * 1
```

Em Haskell, podemos implementar uma função que calcula o fatorial de um número da seguinte forma:

```
fat :: Int -> Int
fat 0 = 1
fat 1 = 1
fat n = n * fat (n-1)
```

A primeira equação diz que o fatorial de 0 é 1. A segunda diz que o fatorial do número 1 é 1. A terceira equação diz que se a função `fat` receber um número `n` (diferente de 1 e 0), calculamos `n * fat (n-1)`. **Exercício:** Implementar uma função recursiva que recebe a base e o expoente e calcula a potência:

```
> potencia 2 3
8
```