Casamento de Padrões

Primeiramente, é importante que os padrões coincidam com o tipo da entrada. Se a entrada é inteira, por exemplo, "3.14" não é um padrão que funcionará.

```
adivinheUmNumero :: Int -> String
adivinheUmNumero 0 = "Aeeeee! 0!"
adivinheUmNumero 73 = "Aeeeee! 73!"
adivinheUmNumero 5 = "Aeeeee! 5!"
adivinheUmNumero 83274 = "Aeeeee! 83274"
```

Na função acima, um inteiro é recebido como entrada e comparado com cada um dos padrões , em ordem, de cima para baixo. Caso coincida com um deles, uma string é retornada e todos os padrões abaixo dele são ignorados, pondo um fim nas comparações.

No entanto, caso os padrões da função não cubram todas as possibilidades de padrões, temos um erro. Nesse caso em particular, uma solução é usar "_" como um dos padrões, que fun ciona como um 'else' ou 'otherwise'.

```
adivinheUmNumero2 :: Int -> String
adivinheUmNumero2 0 = "Aeeeee! 0!"
adivinheUmNumero2 73 = "Aeeeee! 73!"
adivinheUmNumero2 5 = "Aeeeee! 5!"
adivinheUmNumero2 83274 = "Aeeeee! 83274"
adivinheUmNumero2 _ = "Xiiee, errou :'("
```

É importante ter cuidado na utilização de "_" pois, como dito acima, a ordem dos padrões importa. Se for o primeiro dos padrões, por exemplo, a função vai cair nele independente da entrada. Na prática, é isso que ele significa "Independente da entrada, se chegou aqui, é isso". Executando a função abaixo para qualquer inteiro, a saída sempre será a mesma.

Em alguns casos, temos até um warning informando que aquelas linhas de código após o primeiro padrão serão "redundantes" o que de fato faz sentido, já que independente delas, acontecerá o mesmo.

```
adivinheUmNumero3 :: Int -> String
adivinheUmNumero3 _ = "Xiiee, errou :'("
adivinheUmNumero3 0 = "Aeeeee! 0!"
adivinheUmNumero3 73 = "Aeeeee! 73!"
adivinheUmNumero3 5 = "Aeeeee! 5!"
adivinheUmNumero3 83274 = "Aeeeee! 83274"
```