02 - Expressões e Funções

André Rauber Du Bois
dubois@inf.ufpel.edu.br
Computação - CDTec - UFPel

Avaliando Expressões

- Podemos usar o ambiente interativo do Elixir (iex) para avaliar expressões:

```
iex(1)> 33 + 22 * 4
121
iex(2)> String.reverse("Andre")
"erdnA"
iex(3)> "A idade de João é " <> "33 anos."
"A idade de João é 33 anos."
```

- Expressões podem usar operadores, valores e funções pré-definidas

Definindo funções

- A principal abstração em uma linguagem funcional são as funções
- Em Elixir, **funções devem ser definidas em um módulo**
- Por convenção, o **nome do arquivo é sempre o mesmo nome do módulo** sendo definido no arquivo
- Nomes de **módulos** começam com **maiúsculo**
- Nomes de **arquivo** são sempre em **minúsculo**
- O módulo **MeuModulo** deve ficar no arquivo **meu_modulo.ex**

- arquivo: `exemplo.ex`

```
defmodule Exemplo do
 def maior_de_idade(idade) do
   idade >= 18
 end
 def quadrado(x) do
   x * x
 end
 def menor(x,y) do
   cond do
   x >= y -> y
   y > x -> x
   end
 end
end
```

- Definir um módulo: **`defmodule`**
- Definir uma função: **`def`**
- Função pode receber parâmetros ou não
- A **função** sempre **retorna o último valor computado** (não é necessário return)

```
$ iex exemplo.ex

iex(1)> Exemplo.maior_de_idade(33)

true

iex(2)> Exemplo.maior_de_idade(12)

false

iex(3)> Exemplo.quadrado(10)

100

iex(4)> Exemplo.menor(22,33)

22

iex(5)> Exemplo.menor(Exemplo.quadrado(34),Exemplo.quadrado(44))

1156
```

Exercício

- A função **rem** devolve o resto da divisão

```
iex(1)> rem(9,3)
0
iex(2)> rem(10,3)
1
```

- Podemos comparar dois valores com o operador **==**

```
iex(1)> 10 == 10
true
iex(2)> 1 == true
false
```

- Implementar a função par? que retorna **true** se o argumento é par e **false** caso contrário

- Uma solução

```
defmodule Exemplo do
 def par?(num) do
   cond do
     rem(num,2) == 0 -> true
     true            -> false
   end
 end
end
```

- Uma solução melhor:

```
defmodule Exemplo do
  def par?(num) do
    rem(num,2) == 0
  end
end
```

- A construção **when** define uma condição (guarda) para a execução de um caso de uma função

```
defmodule Exemplo do
 def menor(x,y) when x >= y do
   y
 end
 def menor(x,y) when y > x  do
   x
 end
end
```

  - O corpo da função só é executado caso a guarda definida pelo when seja verdadeira. Os casos são testados na mesma ordem das definições

- Exemplo usando **cond**:

```
defmodule Exemplo do
 def valor_absoluto(num) do
   cond do
     num < 0 -> -num
     num == 0 -> 0
     num > 0 -> num
   end
 end
end
```

- Mesmo exemplo usando **when**:

```
defmodule Exemplo do
 def valor_absoluto(num) when num <0 do
   -num
 end
 def valor_absoluto(num) when num == 0 do
   0
 end
 def valor_absoluto(num) when num > 0 do
   num
 end
end
```

- Funções que computam apenas uma expressão, podem usar o comando **do:** ao invés do bloco **do end**

```
defmodule Exemplo do
 def maior_de_idade(idade), do: idade>=18
 def quadrado(x), do: x * x
 def menor(x,y) when x >= y,  do: y
 def menor(x,y) when y > x, do: x
end
```

Operadores

- Operadores lógicos, aritméticos e relacionais são pré-definidos
- Aritméticos: **+** (soma), **-** (subtração), ***** (multiplicação) e */* (divisão)
- Lógicos: **&&** (and), **||** (or) e **!** (negação)
- Relacionais: **>**, **<**, **>=**, **<=**, **==**, **!=**
- Dois **átomos** para os booleanos: **true** e **false**

```
iex(1)> true && true

true

iex(2)> true || true

true

iex(3)> !(true || true)

false
```

- Em expressões booleanas, podemos usar também: **and**, **or** e **not**:

```
iex(1)> true and true

true

iex(2)> not ((true and true) or false)

false
```

Exercício

- Implementar a função **tres_iguais** que recebe três números e devolve **true** caso sejam iguais ou **false** caso contrário

Exercício

- Implementar a função **tres_iguais** que recebe três números e devolve **true** caso sejam iguais ou **false** caso contrário

```
defmodule Exemplo do
 def tres_iguais(a,b,c) do
   a == b && b == c
 end
end
```