Programação Funcional e a Linguagem Elixir

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

Programação Declarativa

- Na **programação imperativa** o programador se preocupa em especificar uma **sequência** de instruções que devem ser executadas em uma certa ordem.
- **Programação declarativa** preocupa-se em especificar **equações** para se resolver o problema sem se preocupar com o fluxo da execução
- **Objetivo:** estudar UMA linguagem de programação declarativa
- **Programação em Lógica** - Prolog
- **Programação Funcional** - Haskell, OCaml, LISP, Scheme, Julia, **Elixir**…

MOTIVAÇÃO PARA PROGRAMAÇÃO FUNCIONAL

- Artigo ***Como se tornar um Hacker*** (How to become a Hacker escrito por Eric Raymond):
- *“Vale a pena aprender LISP pela experiência esclarecedora que o aprendizado proporciona. Essa experiência irá torná-lo um melhor programador para o resto de sua vida, mesmo que você nunca use muito a linguagem LISP”*

Linguagens Funcionais - Lisp

- **Lisp** foi a primeira linguagem funcional. Ela foi descrita em um artigo escrito por McCarthy, do MIT, em 1960
- A função fatorial de um número natural n é o produto de todos os n primeiros números naturais. Ex: fat(3) = 3 * 2 * 1

```
(defun fatorial (n)
  (if (<= n 1)
  1
  (* n (factorial (- n 1)))))
```

Linguagens Funcionais - Haskell

- Criada por um comitê em 1988 para direcionar os estudos da área de programação funcional
- Estaticamente tipada

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

Linguagens Funcionais - Elixir

- Desenvolvida pelo brasileiro José Valim em 2012
- Dinamicamente tipada

```
defmodule Fatorial do
 def fat(0), do: 1
 def fat(n), do: n * fat(n-1)
end
```

O QUE É PROGRAMAÇÃO FUNCIONAL ?

- **Programação funcional** é um estilo de programação que enfatiza a avaliação de expressões. Uma expressão seria composta da aplicação de funções a argumentos
- Uma **Linguagem Funcional** é uma linguagem que estimula ou até obriga o programador a escrever programas pensando apenas em funções e nos valores que elas computam

O que é programação Funcional?

- Funções são os elementos principais desse tipo de linguagem
- Funções podem receber funções como argumentos e uma função pode devolver como resultado uma outra função.
- Estruturas de dados podem conter funções como elementos

Vantagens de Programação Funcional

- Programas são pequenos e com um alto poder de expressão
- Suporta componentes reusáveis de software
- Permite prototipação rápida
- Permite verificação formal dos programas. As funções em uma linguagem funcional são funções matemáticas puras. Uma função que recebe os mesmos argumentos sempre vai devolver a mesma resposta

Exemplo

- Aplicar uma função a todos os elementos de um vetor e depois somar todos os elementos do vetor resultante
- Em C:

```
int quadrado(int x) {
   return x * x;
 }
```

```
void main()
{
    int lista[6] = {1, 2, 3, 4, 5, 6};
    int soma = 0;
    for(int i=0; i<6; i++)
    {
        lista[i] = quadrado (lista[i]);
        soma += lista[i];
    }
    printf("%d",soma);
}
```

Exemplo

- Script Elixir:

```
m = [1,2,3,4,5,6]


r = m
|> Enum.map(fn x -> x * x end)
|> Enum.reduce(0,fn x, y -> x + y end)


IO.puts r
```

```
defmodule Ex do
 def map([], _f), do: []
 def map([h|t], f), do: [f.(h) | map(t,f)]
 def reduce([],acc, _f), do: acc
 def reduce([h|t],acc, f), do: reduce(t,f.(h,acc), f)
end
m = [1,2,3,4,5,6]
r = m
 |> Ex.map(fn x -> x * x end)
 |> Ex.reduce(0,fn x, y -> x + y end)
IO.puts r
```

Programação funcional

- **No programa anterior podemos observar várias características importantes:**
- Uso de **recursão** para **repetições** (percorrer lista)
- **Funções de alta ordem**: funções que recebem funções como argumentos
- **Funções anônimas**
- Operador pipe **|>**: **composição de funções**
- **Imutabilidade dos dados:** update em uma lista gera uma nova lista

Programação Funcional no mundo real

- **Várias abstrações legais disponíveis em linguagens modernas são derivadas das linguagens funcionais:**
- Coleta automática de lixo
- funções anônimas
- List comprehension
- Polimorfismo de tipos/tipos genéricos

Programação funcional no mundo real

- Várias empresas usam linguagens de programação funcional (ICFP 2022)
- **Erlang:** usado para lidar com várias conexões no whats up
- **Haskell:** Modelos financeiros
- **OCaml:** facebook, microsoft, docker (segundo o site do OCaml)
- **Elixir:** Discord
- **Elixir:** https://elixir-lang.org/
- **Elixir:** https://elixir-companies.com/en
- **Elixir:** https://elixir-radar.com/jobs

Programação funcional no mundo real

- Linguagens funcionais são ensinadas nas principais Universidades do mundo
- Muita pesquisa acadêmica
- Vários congressos sobre o tema: ICFP, Haskell Symposium, The Erlang and Elixir Conference, OCaml, Functional and High-Performance Computing, etc.

Elixir

- Linguagem dinâmica e moderna
- Desenvolvida por José Valim em 2012 (ou 2011)
- Sintaxe inspirada em Ruby
- Linguagem de programação funcional
- Dados imutáveis
- **Principal diferencial: Modelo de atores**
  - Programação concorrente e tolerante a falhas
  - Programação distribuída
  - Processos rápidos e com pouco custo
  - Troca de mensagens

Erlang

- Elixir é construída em cima da máquina virtual do **Erlang** (**Beam**)
- Erlang: linguagem desenvolvida na **Ericsson** para programação de switches
- Tese de doutorado do **Joe Armstrong**
- Paradigma: **Modelo de Atores**
  - Processos que não compartilham memória e se comunicam por troca de mensagens assíncronas
- Sistemas **concorrentes e escaláveis** com alta disponibilidade
- **Whats up**
- **Elixir é o Erlang com uma roupa mais moderna**

Curso de Programação Funcional

1. Programação Funcional em Elixir
    a. Tipos básicos e Funções
    b. Funções recursivas
    c. Listas
    d. Tuplas, átomos e pattern matching
    e. Funções de alta ordem
    f. Funções anônimas
    g. Composição de funções
    h. List comprehension
2. Aspectos de Elixir
    a. Maps, Hash Dictionary e Structs
    b. Processos e troca de mensagens
    c. Macros e metaprogramação

Curso de Programação Funcional

- 2 Provas
- 2 Trabalhos

Trabalho 1: Exercícios Resta Um

Trabalho 2: Jogo modo texto

Nota 1: 40% trabalho 1 + 60% prova 1

Nota 2: 50% trabalho 2 + 50% prova 2

Instalação

- Linux

```
> sudo apt install elixir
```

- Windows
  - Download e instalação do Erlang: https://www.erlang.org/downloads.html
  - Download e instalação do Elixir: https://elixir-lang.org/install.html#windows
- Android
  - Pelo termux
- MacOS

```
> brew install elixir
```

Uso - Arquivos

- Arquivo .ex: será compilado para byte-code e depois executado
- Arquivo .exs: script interpretado
- Não existe diferença real, o que vale é a intenção

Uso - Ambiente Interativo: iex

```
$ iex

Erlang/OTP 25 [erts-13.1.3] [source] [64-bit] [smp:8:8] [ds:8:8:10]
[async-threads:1] [jit:ns]


Interactive Elixir (1.14.2) - press Ctrl+C to exit (type h() ENTER for
help)

iex(1)>
```

- **Sair: Ctrl + C**, **Ctrl +C**
- **Sair: Ctrl + C** e depois **a**

```
iex(1)> 13 + 44

57

iex(2)> 45 > 10

true

iex(3)> String.reverse("Subi no onibus")

"subino on ibuS"

iex(4)> String.reverse("Subi no onibus") <> ", é um palíndromo"

"subino on ibuS, é um palíndromo"

iex(5)>
```

- Arquivo: soma.ex

```
defmodule Soma do
  def soma(x,y) do
    x+y
  end
end
```

- Carregar: $ iex soma.ex

```
iex(1)> Soma.soma(4,22)
26
iex(2)> Soma.soma(Soma.soma(4,4), Soma.soma(5,10))
23
iex(3)>
```

Uso - Script

- Arquivo: soma.exs

```
defmodule Soma do
 def soma(x,y) do
   x+y
 end
end
lista = [1,2,3,4,5,6,7]
resultado = Enum.reduce(lista,0,&Soma.soma/2)
IO.puts resultado
```

- Executar script: comando `elixir`

```
$ elixir soma.exs

28
```

- Comando elixir: carrega a máquina virtual, compila o arquivo e executa o código após a declaração do módulo

Uso - Projeto: mix

- Criar um novo projeto

```
$ mix new soma
```

- Cria a pasta soma e várias outras pastas
- Bibliotecas de funções: pasta `lib`
- Pasta lib já contém um arquivo `soma.ex` “vazio”

Uso - mix

- Arquivo: soma.exs

```
defmodule Soma do
 @moduledoc """
 Documentation for `Soma`.
 """
 @doc """
 Hello world.
 ## Examples
     iex> Soma.hello()
     :world
 """
 def hello do
   :world
 end
end
```

Uso: mix

- Substituir o soma.ex pelo soma.ex apresentado anteriormente

  $ mix compile

  Compiling 1 file (.ex)

  Generated soma app

  $ iex -S mix

  Erlang/OTP 25 [erts-13.1.3] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [jit:ns]

  Interactive Elixir (1.14.2) - press Ctrl+C to exit (type h() ENTER for help)

  iiex(1)> Soma.soma(4,5)

  9

Material

- Site do Elixir
  - https://elixir-lang.org/
- Curso no site oficial
  - https://elixir-lang.org/getting-started/introduction.html
- Aprenda Elixir: coleção de recursos
  - https://github.com/oguhpereira/aprenda-elixir
- Livro: Joy of Elixir:
  - https://joyofelixir.com/
- Elixir School (em português)
  - https://elixirschool.com/pt/
- Em português
  - http://victorolinasc.github.io/elixir_dojo/dojo.html
- Cursos do professor Adolfo Neto (UTFPR)

Livros

- **Introducing Elixir, getting started in functional programming.** Simon St. Laurent e J. David Eisenberg
- **Elixir in Action**. Saša Jurić
- **Programming Elixir 1.6, Functional |> Concurrent |> Pragmatic |> Fun.** Dave Thomas
- **Learn Functional Programming with Elixir, New Foundations for a New World.** Ulisses Almeida
- **Metaprogramming Elixir, Write Less Code, Get More Done (and Have Fun!).** Chris McCord

Nesta semana

- Instalar o Elixir
- Testar os exemplos desta aula
- Ler o capítulo básico do Elixir School testar os exemplos
  https://elixirschool.com/pt/lessons/basics/basics