CONGRESSO CATARINENSE DE
CIÊNCIA DA COMPUTAÇÃO
2022. Ano 9, Vol. 9. ISSN: 2319-0655

LINGUAGEM FUNCIONAL ELIXIR

Cristian Gabriel Butzke[1], Júlio Werner Zanatta Koepsel[2], Daniel Gomes Soares[3]
[1]Estudante do curso de Bacharelado em Ciências da Computação do Instituto Federal Catarinense, IFC, Rio do Sul - SC
[2]Estudante do curso de Bacharelado em Ciências da Computação do Instituto Federal Catarinense, IFC, Rio do Sul - SC
[3]Professor do curso de Bacharelado em Ciências da Computação do Instituto Federal Catarinense, IFC, Rio do Sul - SC
cristian.btzk@gmail.com, julio.koepsel@gmail.com, daniel.soares@ifc.edu.br

Abstract. *The Lambda Calculus, created by Alonzo Church, served as a basis for the creation of several functional paradigm languages, the first being LISP. In 2011, José Valim started the development of a project named Elixir, a functional programming language based on Erlang, running on the Erlang Virtual Machine. Having worked for a long time with the Ruby language, considered by many to be an extremely productive language, Valim knew how to take advantage of its features and add them to Elixir, thus creating a simple, scalable language suitable for concurrent programming. This article is a study of the history and technical aspects of the Elixir language.*
Key-words: *Elixir; Functional Programming; Concurrence.*

Resumo. *O Cálculo Lambda, criado por Alonzo Church, serviu como base para a criação de diversas linguagens do paradigma funcional, sendo a primeira delas o LISP. Em 2011, José Valim iniciou o desenvolvimento de um projeto que recebeu o nome de Elixir, uma linguagem de programação funcional baseada em Erlang, executada sobre a Máquina Virtual Erlang. Por ter trabalhado por um grande período com a linguagem Ruby, considerada por muitos uma linguagem extremamente produtiva, Valim soube aproveitar as suas características e agregá-las ao Elixir, criando assim uma linguagem simples, escalável e própria para programação concorrente. O presente artigo é um estudo da história e aspectos técnicos da linguagem Elixir.*
Palavras-chave: *Elixir; Programação Funcional; Concorrência.*

1. Introdução

Com a introdução do formalismo matemático que estuda funções computáveis, denominado Cálculo Lambda, por Alonzo Church, um novo paradigma de programação foi logo introduzido, utilizando como base esse sistema. Um breve exemplo de como as linguagens de programação implementam o cálculo Lambda, são as funções anônimas (STANFORD, 2012).

A criação do LISP, por John McCarthy, foi o marco inicial das linguagens funcionais. Além disso, a linguagem introduziu vários conceitos que são utilizados ainda hoje na programação (MCCARTHY, 1979).

O presente artigo busca explorar a história e os detalhes da linguagem funcional Elixir, demonstrando aspectos históricos e técnicos da linguagem. Além disso, serão

demonstrados os recursos básicos da linguagem, com implementações e exemplos práticos.

2. Metodologia

Para a elaboração do artigo, realizou-se uma pesquisa exploratória, a fim de compreender a história da linguagem Elixir e seus predecessores. A pesquisa ocorreu em diversas fontes, principalmente artigos e materiais de confiabilidade sobre a linguagem, além de dados disponíveis no *website* oficial da linguagem elaborados pelo seu próprio criador.

Após a coleta de dados, estes foram organizados e avaliados, onde verificou-se sua importância no artigo; Além disso, o material consultado foi verificado em mais de uma fonte de pesquisa, para garantir sua veracidade.

3. Resultados e Discussões

3.1. Programação Funcional

A programação funcional é um paradigma de programação com foco no uso de funções para o desenvolvimento de sistemas complexos. Esse paradigma aborda a construção de *software* através da composição de funções puras, evitando compartilhamento de estados, dados mutáveis e efeitos colaterais, utilizando um estilo de programação declarativo ao invés do imperativo (NASCIMENTO, 2019).

Funções são fundamentais para a organização do código e estão presentes em várias linguagens de programação. Muitas linguagens não estão limitadas a somente um paradigma e aceitam diferentes abordagens como programação funcional, orientada a objetos, procedural, etc (TYSON, 2021).

Em termos de vantagens, o código em programação funcional tende a ser mais objetivo do que os outros tipos de paradigmas. A programação funcional possui maior flexibilidade, notação concisa, semântica simples e maior facilidade nos testes e na busca por erros, facilitando o trabalho dos desenvolvedores (ROVEDA, 2021).

A programação funcional possui alguns conceitos fundamentais, sendo eles: composição de função, funções puras, imutabilidade, efeito colateral e estado compartilhado. Composição de função: criação de uma função através da composição de outras. Funções puras: funções que dado um parâmetro de entrada (que não seja compartilhado) sempre vão retornar a mesma saída, sem causar efeitos colaterais. Imutabilidade: uma vez que uma variável recebe um valor, vai possuir esse valor para sempre, ou quando um objeto for criado, ele não poderá ser modificado. Efeito colateral: uma modificação no estado da aplicação que seja percebida fora do destino da função chamada. Uma função possui efeito colateral quando altera o estado fora do seu contexto local. Estado compartilhado: uma variável ou objeto que existe em um escopo

compartilhado. A ocorrência ou implementação desses conceitos depende da linguagem utilizada (ROVEDA, 2021).

3.2. Erlang

Para compreender o nascimento do Elixir, é essencial compreender o que é Erlang, e qual a relação entre ambas. Erlang é uma linguagem de programação criada na década de 80, mais precisamente, no ano de 1986. Seu objetivo original era resolver problemas de confiabilidade em conexões em centrais de telecomunicações. Um dos objetivos era criar uma linguagem que possibilitasse a criação de programas que não pudessem ser interrompidos, ou seja, deveriam "funcionar para sempre" (MOSTOVOY, 2020).

Dessa maneira, nasce Erlang, uma linguagem de programação funcional com foco em concorrência e disponibilidade contínua. Dentre as inovações que surgiram com a sua criação, destacam-se: *Green Threads*, que são *threads* escalonadas por uma máquina virtual, e não pelo sistema operacional; AMQP, um protocolo influenciado pela troca de mensagens entre processos realizada no Erlang; Continuous Delivery, uma prática onde as atualizações e manutenções devem ser realizadas de forma rápida e segura, nascida pelo fato de sistemas de telefonia necessitarem estar em execução a todo momento (MOSTOVOY, 2020).

A linguagem Erlang é utilizada principalmente em *Chat Apps*, serviços distribuídos de alta performance e sistemas de mensageria (MOSTOVOY, 2020).

3.3. Surge o Elixir

Atualmente, Erlang possui uma popularidade não muito alta, em contrapartida a uma nova linguagem que vem ganhando cada vez mais atenção, denominada Elixir.

O início da linguagem Elixir se deu no ano de 2012, como um projeto de pesquisa e desenvolvimento interno na empresa Plataformatec. Seu criador, José Valim, tinha como objetivo criar uma linguagem produtiva e extensível, de forma que pudesse ser usada no desenvolvimento de *softwares* de forma manutenível e confiável (ELIXIR, 2022).

A linguagem é executada sobre a Máquina Virtual Erlang, que proporciona uma fundação escalável e tolerante a erros. Elixir foi projetado para aproveitar essa fundação sem custos de performance e contribuir para um ecossistema mais amplo (ELIXIR, 2022).

Valim foi, antes de desenvolver o Elixir, desenvolvedor Ruby. Desta maneira, as linguagens possuem algumas similaridades. Durante seus trabalhos em Ruby, a dificuldade de realizar concorrência o fez buscar maneiras de resolver

esse problema, e se interessou pela máquina virtual do Erlang (SOPPA, 2022).

Ruby é considerada uma linguagem produtiva. Quanto mais rápido a aplicação for criada e realizar seu papel, menor é o custo envolvido. Por essa maneira, várias empresas resolveram adotá-la. Entretanto, essa produtividade não está presente no Erlang. Valim compreendeu essas características, e Elixir é uma linguagem produtiva e facilmente escalável (SOPPA, 2022).

O código fonte da linguagem está sobre a licença Apache 2.0, e é mantido por um grupo de desenvolvedores conhecidos como *Elixir Core Team*, formado por Aleksei Magusev, Andrea Leopardi, Eric Meadows-Jönsson, Fernando Tapia Rico, e o seu criador, José Valim (ELIXIR, 2022).

O início do desenvolvimento se deu em janeiro de 2011, e a versão 1.0 da linguagem foi lançada em Setembro de 2012 (VALIM, 2014).

3.4. Características Técnicas e Recursos da Linguagem

3.4.1 Tipos

A linguagem Elixir é formada por dados de vários tipos. Valores do tipo inteiro (Integer), podem ser expressados em base decimal, binária, octal e hexadecimal. Valores armazenados com ponto flutuante (Float) necessitam de um ponto seguido de ao menos um número. Além disso, suportam *e* para notação científica. Em Elixir, valores do tipo *float* possuem precisão dupla de 64 bits. Existem também valores booleanos: verdadeiro (*true*) e falso (*false*). Átomos são valores constantes cujo valor é o seu próprio nome. Em outras linguagens, recebem o nome de Símbolos. Os valores booleanos *true* e *false*, por exemplo, são átomos, e a comparação *true == :true* é válida. *Strings* são cadeias de caracteres utilizadas geralmente para simbolizar palavras, delimitadas na linguagem por aspas duplas e utilizam encode UTF-8. Tuplas são definidas através de um par de chaves, e podem conter qualquer tipo. Tuplas são armazenadas contiguamente na memória. Portanto, podem ser acessadas através de um índice, que inicia em 0. Listas são estruturas de dados que armazenam um conjunto de valores. Em Elixir, listas são definidas através do uso de colchetes, e seus valores podem ser de qualquer tipo. Duas listas podem ser concatenadas ou subtraídas utilizando - - ou ++. Como as estruturas de dados na linguagem são imutáveis, o retorno de uma adição ou subtração gera uma nova lista. Listas, em Elixir, são Listas Encadeadas. Logo, uma operação de adicionar um valor ao fim de uma lista leva mais passos do que adicionar ao seu início. A figura 1 exibe alguns tipos de dados na linguagem Elixir (ELIXIR, 2022).

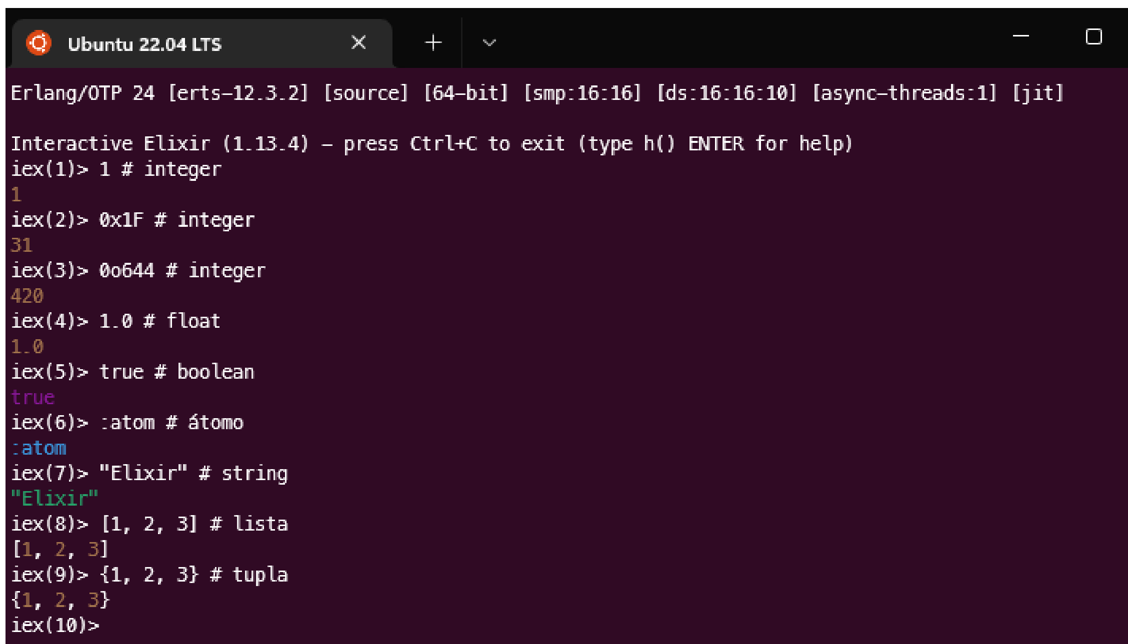

Figura 1 – Tipos de dados em Elixir

3.4.2 Operadores

Os quatro operadores matemáticos básicos, em Elixir são "+", "-", "*" e "/", que correspondem a adição, subtração, multiplicação e divisão, respectivamente. Além disso, existem três operadores booleanos: *or*, *and* e *not*. Esses operadores recebem como primeiro argumento uma expressão booleana. Outros três operadores booleanos existem: ||, && e !, que aceitam argumentos de qualquer tipo. Operadores de comparação também são disponibilizados. ==, !=, ===, !==, <=, >=, < e > são operadores de comparação que significam, respectivamente, igualdade de valores, negação, igualdade de valor e tipo, negação, menor ou igual a, maior ou igual a, menor que e maior que. O operador "<>" é utilizado para concatenar valores do tipo *String*. A figura 2 exibe alguns operadores da linguagem (ELIXIR, 2022).

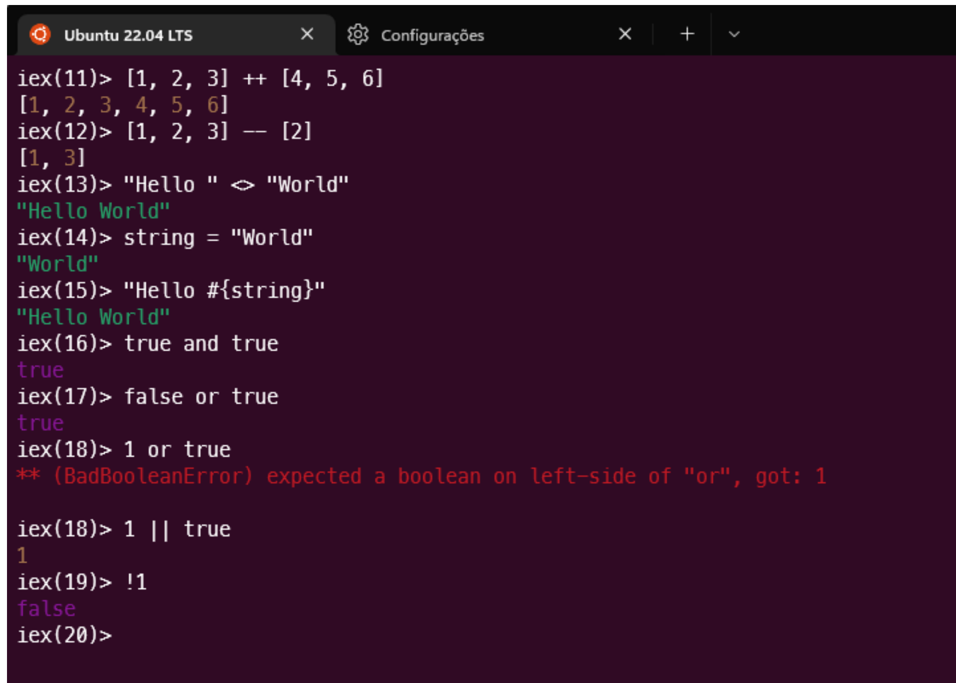

Figura 2 – Operadores

3.4.3 Pattern Matching

Em Elixir, o operador "=" é chamado de *Match Operator*, e tenta combinar os valores dos dois lados da operação. Variáveis podem ser atribuídas quando declaradas do lado esquerdo de uma expressão com o operador *Match*. Além disso, esse operador pode ser utilizado para desestruturar tipos de dados mais complexos. Por exemplo, a figura 3 mostra o uso do *Pattern Matching*. Nas linhas 1 e 5 foram realizadas atribuições de variáveis. É possível notar que ao verificar "10 = i" um erro foi exibido, pois não houve combinação entre o valor 10 e a variável i, que possui valor 2. É possível visualizar ainda a desestruturação de uma tupla em três variáveis (a, b e c) e também de uma lista (d, e e f,) (ELIXIR, 2022).

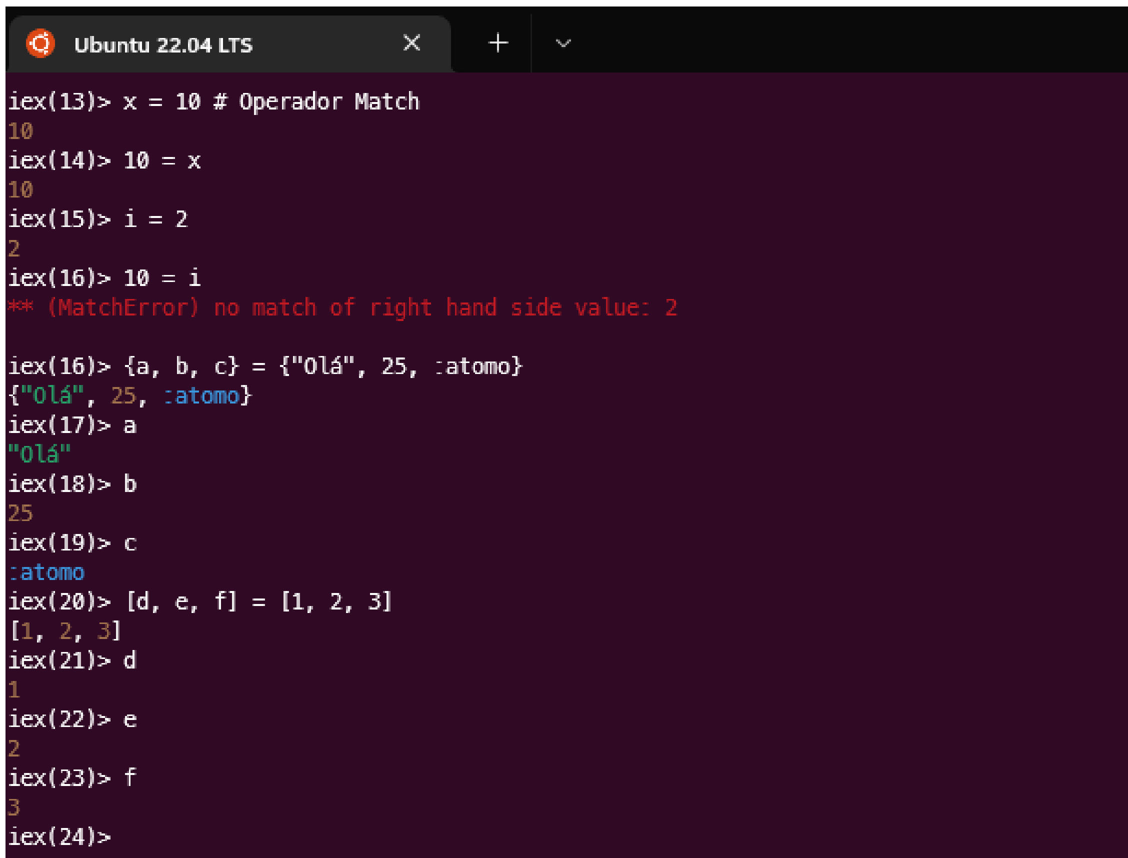

Figura 3 – Pattern Matching

3.4.4 Estruturas de Controle

Em Elixir, algumas estruturas de controle podem ser utilizadas. *Case* é uma estrutura que permite a comparação de um valor com vários padrões, até encontrar uma combinação válida, e então executa o código referente. É Possível também atribuir uma variável enquanto realiza a comparação, assim como em Pattern Matching (ELIXIR, 2022).

Cond permite fazer comparações diferentes, e busca a primeira que satisfaça a condição. É equivalente a uma estrutura *else if* presente em várias linguagens imperativas (ELIXIR, 2022).

If/else verifica se uma condição é verdadeira, e executa o código correspondente, enquanto *Unless/Else* executa um trecho caso a condição estabelecida seja falsa. A figura 4 mostra um exemplo das diferentes estruturas de controle apresentadas (ELIXIR, 2022).

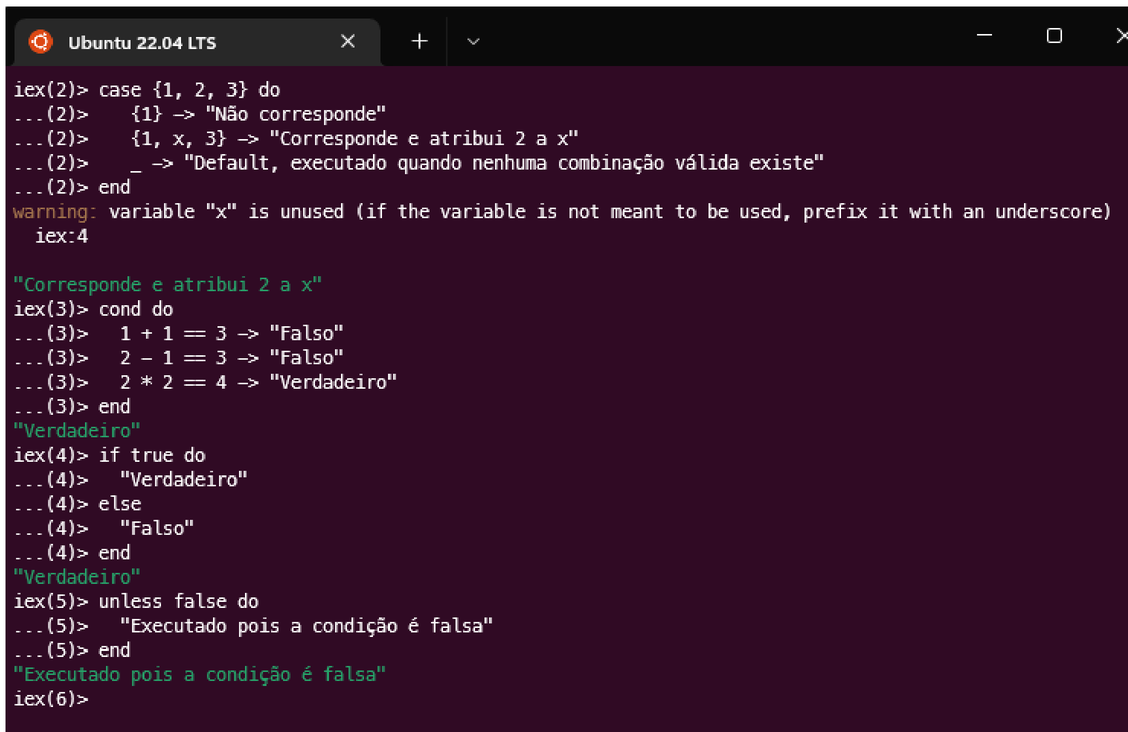

Figura 4 – Estruturas de controle

3.4.5 Módulos e Funções Nomeadas

É possível agrupar várias funções em módulos. Um módulo é criado utilizando a palavra *defmodule*. As funções dentro de um módulo são definidas através de *def*. Funções também podem ser declaradas de forma privada, de modo que só podem ser chamadas localmente (ELIXIR, 2022).

Outro recurso da linguagem é a recursão. Linguagens funcionais não implementam *loops*, como linguagens imperativas. Dessa forma, é necessário utilizar de um recurso chamado recursividade, que consiste em chamar a função dentro do seu próprio código. A função recursiva será executada até que uma condição interrompa o fluxo. Uma função possui diversas cláusulas (declarações). Uma das cláusulas é executada quando os argumentos passados correspondem ao argumento da cláusula, e a condição referente é satisfeita. A figura 5 mostra a declaração de um módulo chamado "Recursao", que possui uma função chamada repetir, que recebe uma mensagem e exibe um certo número de vezes. É notável ainda a saída de um erro, que ocorre por não haver uma cláusula que aceita a função repetir tendo como argumento o número 0 como quantidade de repetições (ELIXIR, 2022).

```
Ubuntu 22.04 LTS        Configurações
iex(4)> defmodule Recursao do
...(4)>   def repetir(msg, n) when n > 0 do
...(4)>     IO.puts(msg)
...(4)>     repetir(msg, n - 1)
...(4)>   end
...(4)> end
{:module, Recursao,
 <<70, 79, 82, 49, 0, 0, 5, 144, 66, 69, 65, 77, 65, 116, 85, 56, 0, 0, 0, 159,
   0, 0, 0, 17, 15, 69, 108, 105, 120, 105, 114, 46, 82, 101, 99, 117, 114, 115,
   97, 111, 8, 95, 95, 105, 110, 102, 111, ...>>, {:repetir, 2}}
iex(5)> Re
Record      Recursao    Regex       Registry
iex(5)> Recursao.repetir
repetir/2
iex(5)> Recursao.repetir("Olá ", 3)
Olá
Olá
Olá
** (FunctionClauseError) no function clause matching in Recursao.repetir/2

    The following arguments were given to Recursao.repetir/2:

        # 1
        "Olá "

        # 2
        0

    iex:5: Recursao.repetir/2
```

Figura 5 – Módulos, Funções e Recursão

4. Conclusão

É possível concluir, portanto, que Elixir é uma linguagem de programação criada sobre a antiga linguagem Erlang, que leva consigo influência de outras linguagens como, por exemplo, Ruby.

Seu objetivo é proporcionar um ambiente simples e escalável, de modo que a criação de softwares seja extremamente simples e minimize os custos de produção. Por ser executada na Máquina Virtual Erlang, utiliza recursos como tolerância a erros e outras fundações do Erlang, sem custos de performance.

Os detalhes básicos da linguagem podem ser encontrados na documentação oficial. Recursos como *Pattern Matching* podem ser extremamente úteis em diversos casos de desenvolvimento.

Referências

ELIXIR. Development. **Elixir**, 2022. Disponível em: https://elixir-lang.org/development.html. Acessado em: 10 mai. 2022.

MCCARTHY, John. History of Lisp. Stanford University, 1979. Disponível em: http://jmc.stanford.edu/articles/lisp/lisp.pdf. Acessado em: 12 mai. 2022.

MOSTOVOY, John. Game of Phones: History of Erlang and Elixir. **Serokell**, 2020. Disponível em: https://serokell.io/blog/history-of-erlang-and-elixir. Acessado em: 07 mai. 2022.

NASCIMENTO, Felipe. Programação funcional: O que é?. **Alura**, 2019. Disponível em: https://www.alura.com.br/artigos/programacao-funcional-o-que-e. Acessado em: 14 mai. 2022.

ROVEDA, Ugo. Programação funcional: O que é e seus principais conceitos. **Kenzie**, 2021. Disponível em: https://kenzie.com.br/blog/programacao-funcional/. Acessado em: 14 mai. 2022.

SOPPA, Szymon. The little story of Elixir programming language. **Curiosum**, 2022. Disponível em: https://curiosum.com/blog/story-of-elixir-programming-language. Acessado em: 08 mai. 2022.

STANFORD. The Lambda Calculus, 2012. Disponível em: https://plato.stanford.edu/entries/lambda-calculus/#Int. Acessado em: 12 mai. 2022.

TYSON, Matthew. What is functional programming? A practical guide. 2021. Disponível em: https://www.infoworld.com/article/3613715/what-is-functional-programming-a-practical-guide.html. Acessado em: 14 mai. 2022.

VALIM, José. Elixir v1.0 released. **Elixir**, 2014. Disponível em: https://elixir-lang.org/blog/2014/09/18/elixir-v1-0-0-released/. Acessado em: 10 mai. 2022.