Universidade Federal do Espírito Santo
Centro Tecnológico
Departamento de Informática

Erlang

Luiz Carlos Passamani Filho

Vitoria, 2015

Agenda

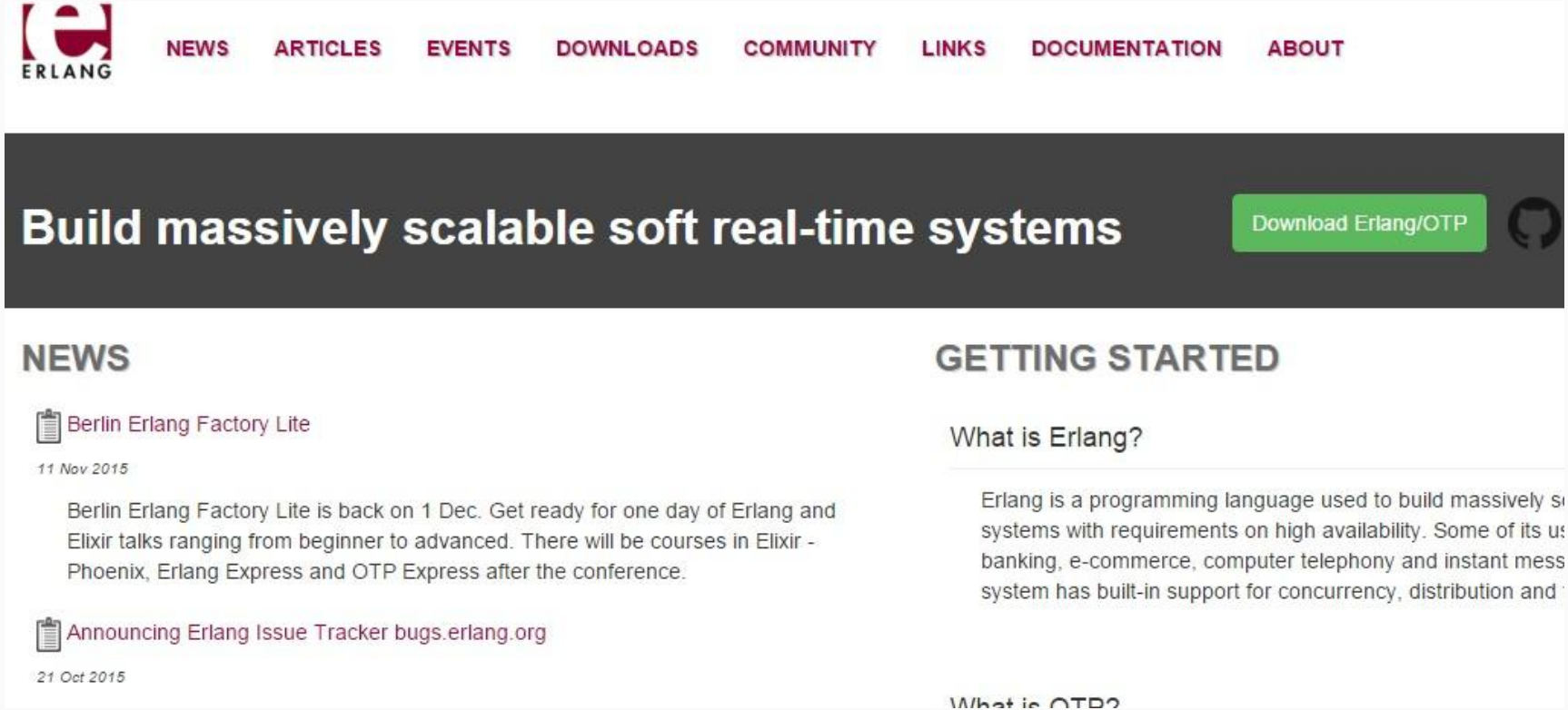
- **Introdução**
- Amarrações
- Valores e tipos de dados
- Variáveis e constantes
- Gerenciamento de Memória
- Expressões e comandos
- Modularização
- Polimorfismo
- Exceções
- Concorrência
- Avaliação da Linguagem
- Referências Bibliográficas

Introdução

- Originalmente era uma linguagem proprietária da Ericsson, mas foi lançada em código aberto em 1998;
- Linguagem funcional;
- A implementação da Ericsson executa um código interpretado em uma máquina virtual, mas também inclui um compilador para código nativo;
- Foi desenvolvida para suportar aplicações distribuídas, tolerantes a falhas e a serem executadas em um ambiente de tempo real e ininterrupto;
- Suporta nativamente hot swapping, de forma que o código pode ser modificado sem a parada do sistema;
- Atualmente, Erlang é utilizada no desenvolvimento de sistemas como o CouchDB e o chat do Facebook.

Introdução

Link para downlaod: http://www.erlang.org/



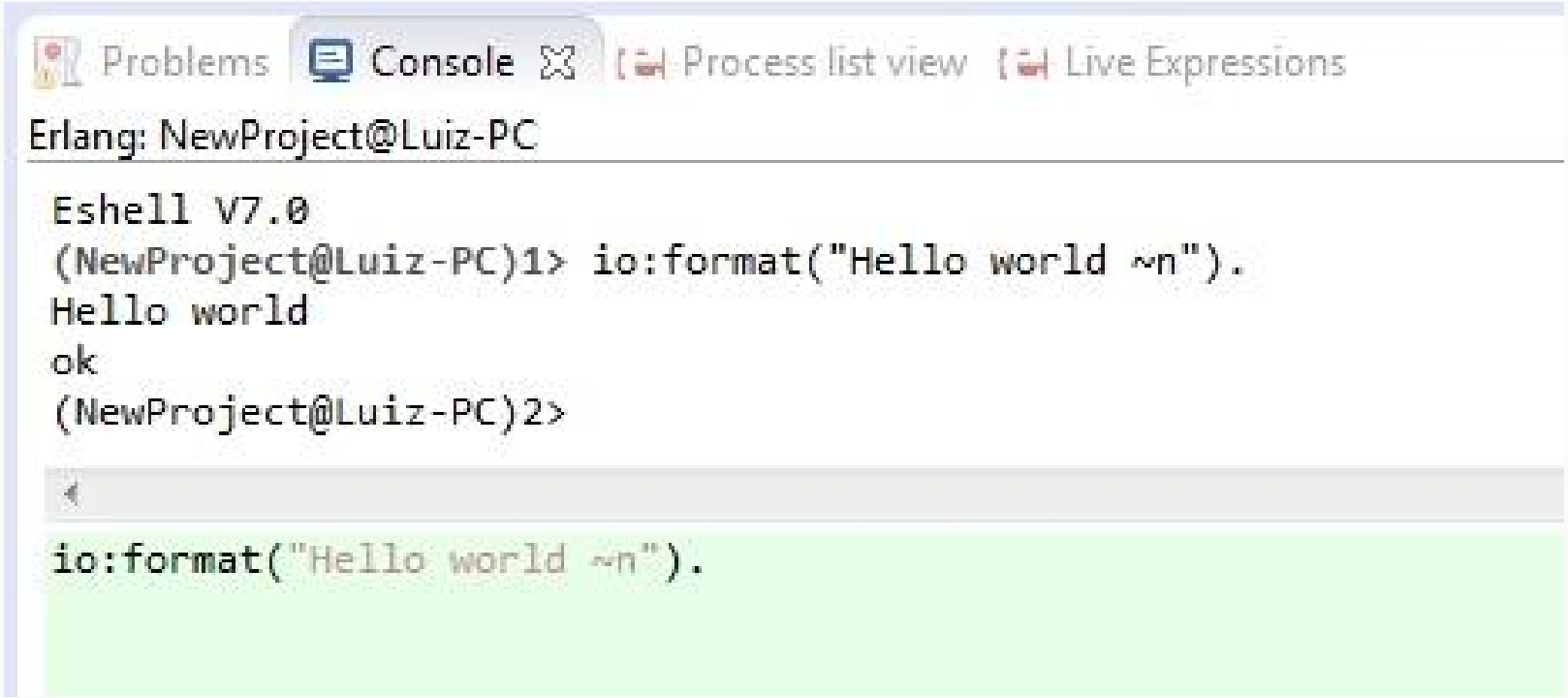
Ambientes de Desenvolvimento

- Erlang Shell
  - Desenvolvido pela Ericsson
- ErlyBird
  - DE baseado no NetBeans
- Erlide
  - Plug-in para o Eclipse
  - Instalação:

http://www.cin.ufpe.br/~dclal/arquivos/instalacao.pdf

Introdução

Primeiro programa:



Agenda

- ✔ Introdução
- ➢ **Amarrações**
- ➢ Valores e tipos de dados
- ➢ Variáveis e constantes
- ➢ Gerenciamento de Memória
- ➢ Expressões e comandos
- ➢ Modularização
- ➢ Polimorfismo
- ➢ Exceções
- ➢ Concorrência
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Amarrações

- Amarração estatica, e depois de declarado o valor da variavél, ela nao pode ser alterada.

```
39> M = 5.
5
40> M = 6.
** exception error: no match of right hand side value 6
41> M = M + 1.
** exception error: no match of right hand side value 6
42> N = M + 1.
6
```

Identificadores

- Sem limites de tamanho;
- Não é case-sensitive;
- Variáveis devem começar com letra maiuscula ou underline;
- Funções devem começar com letra minuscula.

```
_123 = 10,
V@raiável = 10,
Teste_1@2 = 10.
```

Palavras Reservadas

after	and	*andalso*	band	begin	bnot	bor
bsl	bsr	bxor	case	catch	cond	div
end	fun	if	let	not	of	or
orelse	receive	rem	try	when	xor	%

Escopo das variáveis

- Em Erlang as variáveis tem escopo estatico definido por blocos delimitados por (**->** e **.**).
  - Ex:

    funcao(<**parametros**>) **->**

    código1,

    .
- Variáveis definidas dentro de blocos nao são visíveis fora do bloco.
- Para ter uma variável com visibilidade global usa-se *-define*.

Agenda

- ✔ Introdução
- ✔ Amarrações
- ➢ **Valores e tipos de dados**
- ➢ Variáveis e constantes
- ➢ Gerenciamento de Memória
- ➢ Expressões e comandos
- ➢ Modularização
- ➢ Polimorfismo
- ➢ Exceções
- ➢ Concorrência
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Tipos de Dados

Tipos primitivos:

- Inteiros: 10, -234, 16#AB10F, 2#1101101101
- Float: 17.368, -56.654, 12.34E-10
- Char: O tipo Char é, em Erlang, nada mais que um código inteiro da tabela ASCII. Tipos Char são representados por "$". Ex: $a, $s, $d.
- Átomo:
  - Tamanho indefinido
  - São constantes literais;
  - O seu valor é o próprio átomo;
  - Começam com uma letra minúscula;
  - Qualquer código de caractere é permitido entre aspas;
  - Somente operações de comparação é permitida entre átomos;
  - Utilizado para qualificar ou expressar os dados.
- Bool: não existe oficialmente na linguagem, somente em forma de átomos.1<2 retornaria true, assim como 23==17 retornaria false.

Tipos de Dados

Tipos compostos:

- Tuplas
  - São usadas para armazenar um número fixo de elementos;
  - São permitidas tuplas de qualquer tamanho;
  - Podem armazenar dados de diversos tipos;
  - Ex: {1, "pessoa", {carro,moto}, 1.4}.
- Listas
  - São usadas para armazenar um número fixo de elementos;
  - Apenas elementos do mesmo tipo;
  - Ex: [1,2,3] ,[2.55, 3.14, 1.47].

Tipos de Dados

Tipos compostos:

- Strings
  - São basicamente listas de números inteiros, que representam os caracteres;
  - Ex: "hello" == [$h, $e, $l, $l, $o] == [104, 101, 108, 108, 111].
- Record:
  - Semelhantes aos structs em C. Para criar um tipo record deve informar o nome do e das informações composta por ele.
  - Ex. declaração: -record(pessoa, {nome = "default", numero = "default"}).

Manipulação record

- Records não são um tipo composto oficial da linguagem, pois quando o código é pré-compilado ele é transformado em tuplas:
  - Ex. criação: #pessoa{nome = luiz, numero = 999}.
  - Retorno da Shell: {pessoa, fulano, 44}

- Acessar uma informação do campo do record:
  - **<Variavel>#<nome-record>.<campo-do-record>**.
    - Ex: Cliente1#pessoa.nome.

Manipulação record

- Atualizar um campo do record:
  - **<Variavel>#<nome-record>{<campo-do-record> = <valo>}** .
    - Ex: X#pessoa{nome = pedro}.

- Acessar record em uma lista de records:
  - lists:keysearch(**<chave>**, **<campo>**, **<lista-record>**).
    - EX: lists:keysearch(pedro, #person.name, ListaRecord).

Agenda

- ✔ Introdução
- ✔ Amarrações
- ✔ Valores e tipos de dados
- ➢ **Variáveis e constantes**
- ➢ Gerenciamento de Memória
- ➢ Expressões e comandos
- ➢ Modularização
- ➢ Polimorfismo
- ➢ Exceções
- ➢ Concorrência
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Propriedades de variáveis

- São iniciadas por letras maiúsculas ou underline;
- Depois de inicializadas nao podem ser alteradas;
- Não tem manipulação de ponteiros;
- Não é necessario declarar variáveis;
- Não tem variável global.

Declaração de constantes

- Não existe declaração de constantes, mas todas variaveis se comportam como constantes, pois nao podem ser alteradas.
- Exemplo declaração de define:

```
15  -define(N, 123).
16  -define(M, "what").
17  -define(SQUARED (X), X*X).
18
19  showConstants() ->
20      io:format("N = ~p ~n", [?N]),
21      io:format("M = ~p ~n", [?M]),
22      io:format("~p ~n", [?SQUARED(5)]).
```

Problems | Console | Process list view | Live Expressions

Erlang: NewProject@Luiz-PC

```
(NewProject@Luiz-PC)5> teste:showConstants().
N = 123
M = "what"
25
ok
(NewProject@Luiz-PC)6>
```

I/O

- Exemplo manipulação de arquivos:

```
16 leArquivo()->
17 io:format("~n"),
18 le_arquivo("clientes.txt").
19
20
21 le_arquivo(NomeDoArquivo)->
22 {ok, Binario} = file:read_file(NomeDoArquivo),
23 ListaDeNomes = string:tokens(erlang:binary_to_list(Binario), ";\r\n"),
24
25 X = erlang:list_to_tuple(ListaDeNomes),
```

Problems | Console | Process list view | Live Expressions

Erlang: NewProject@Luiz-PC

```
(NewProject@Luiz-PC)32> lerArquivo:leArquivo().

{"Luiz Carlos","998495400","Joao Paulo","123456","Pedro",
 "99999"}
(NewProject@Luiz-PC)33>
```

Gerenciamento de memória

- Em Erlang, o gerenciamento de memoria é feita pelo coletor de lixo;
- O coletor de lixo faz uma copia da memoria e em seguida os ponteiros ativos sao copiados para a nova memoria alocada e os inativos sao ignorados.
- Os Átomos são armazenados em uma área de dados separada e nao podem ser excluidos ate o fim da execução do programa.

Agenda

- ✓ Introdução
- ✓ Amarrações
- ✓ Valores e tipos de dados
- ✓ Variáveis e constantes
- ➢ **Expressões e comandos**
- ➢ Modularização
- ➢ Polimorfismo
- ➢ Exceções
- ➢ Concorrência
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Operadores

op	*Description*
not	unary logical not
and	logical and
or	logical or
xor	logical xor
andalso	short-circuit and
orelse	short-circuit or

Lógicos

op	*Description*
==	equal to
/=	not equal to
=<	less than or equal to
<	less than
>=	greater than or equal to
>	greater than
=:=	exactly equal to
=/=	exactly not equal to

Relacionais

op	*Description*	*Argument type*
/	floating point division	number
bnot	unary bitwise not	integer
div	integer division	integer
rem	integer remainder of X/Y	integer
band	bitwise and	integer
bor	bitwise or	integer
bxor	arithmetic bitwise xor	integer
bsl	arithmetic bitshift left	integer
bsr	bitshift right	integer

Aritméticos

Operadores

- Operadores de manipulação de listas:

```
(NewProject@Luiz-PC)50> A = [1,2,3].
[1,2,3]
(NewProject@Luiz-PC)51> B = [3,4,5].
[3,4,5]
(NewProject@Luiz-PC)52> A++B.
[1,2,3,3,4,5]
(NewProject@Luiz-PC)53> A--B.
[1,2]
(NewProject@Luiz-PC)54>
```

```
(NewProject@Luiz-PC)35> [X || X <- [1,2,3,4,5,6,7,8], X < 5, X > 2].
[3,4]
(NewProject@Luiz-PC)36> [{X,Y} || X <- [1,2], Y <- [5,6]].
[{1,5},{1,6},{2,5},{2,6}]
(NewProject@Luiz-PC)37>
```

Definição de função

- Devem sempre começar com letra minuscula;
- As funções devem ser declaradas no inicio do programa: -export ([<**nome_da_função**>/<**numero_de_parâmetros**>]);
- Possuem passagem de parâmetro por valor.

Definição de função

- Para chamar uma função, utilizamos a sintaxe:
  - ***<nome-do-módulo> : <nome-da-função>(<parâmetros>).***

```
1  -module(fat).
2
3  -export([fatorial/1]).
4
5  fatorial(N) when N > 0 ->
6      N * fatorial(N - 1);
7  fatorial(0) ->
8      1.
9
```

Problems | Console | Process list view | Live Expressions

Erlang: NewProject@Luiz-PC

```
(NewProject@Luiz-PC)3> cd("C:/Users/Luiz Carlos/workspace/NewProject/src").
C:/Users/Luiz Carlos/workspace/NewProject/src
ok
(NewProject@Luiz-PC)4> c(fat).
{ok,fat}
(NewProject@Luiz-PC)5> fat:fatorial(5).
120
(NewProject@Luiz-PC)6>
```

Expressões condicionais

- A expressão **case of**: (Similar a um switch)

```
1  -module(fat).
2
3  -export([fatorial/1,condicional/1]).
4
5  condicional(Cidade) ->
6      case Cidade of
7          "Vitoria" -> vix;
8          "Sao Paulo" -> sampa;
9          _ -> nenhuma
10     end.
```

Problems | Console | Process list view | Live Expressions

Erlang: NewProject@Luiz-PC

```
(NewProject@Luiz-PC)15> fat:condicional("Vitoria").
vix
(NewProject@Luiz-PC)16> fat:condicional("Serra").
nenhuma
```

Expressões condicionais

- A expressão **if** sempre deve ter um resultado da condição **true:**

```
5  condicional(X,Y) ->
6      if
7          X > Y -> maior;
8          Y > X -> menor;
9          true -> iguais
10     end.
```

Problems | Console | Process list view | Live Expressions

Erlang: NewProject@Luiz-PC

```
(NewProject@Luiz-PC)19> fat:condicional(1,1).
iguais
```

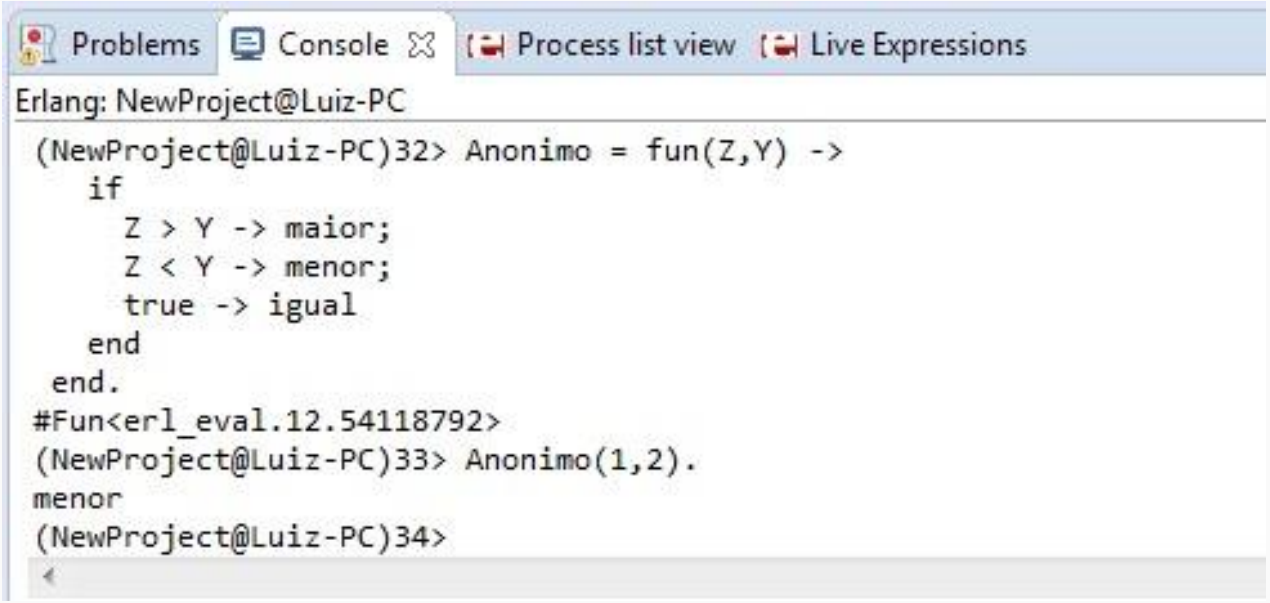
Recursão

- Como nao Erlang nao possui estruturas de repetições(For, While), utiliza-se a recursao:

```
38  calcula(X,Valor,Pedido,Lanches) when X > 0 ->
39      NomeLanche = erlang:element(X,Pedido),
40      Lanche = lists:keysearch(NomeLanche,#lanche.nome,Lanches),
41      RetornaRecordLanche = erlang:element(2,Lanche),
42      calcula(X-1,Valor + RetornaRecordLanche#lanche.valor,Pedido,Lanches);
43      calcula(0,Valor,Pedido,Lanches) ->
44          Valor.
45
```

Função Anônima

- Sintaxe básica:
  - **fun(<lista-de-parâmetros>) <corpo-da-função> end.**



Agenda

- ✓ Introdução
- ✓ Amarrações
- ✓ Valores e tipos de dados
- ✓ Variáveis e constantes
- ✓ Expressões e comandos
- ➢ **Modularização**
- ➢ Polimorfismo
- ➢ Exceções
- ➢ Concorrência
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Modularização

- Erlang permite criação de funções por parte do usuário;
- Funções podem ser declaradas em arquivos separados(Módulos);
  - Ex: ***<nome-do-módulo> : <nome-da-função>(<parâmetros>).***
- Funções podem ser utilizadas em pacotes(package) diferentes;
  - Ex: cd("**<diretorio-do-pacote>**").

- Parâmetros são sempre passados por valor;
- Momento de passagem de parâmetro não é preguiçoso.

Modulos

- Alguns modulos que fazem parte da biblioteca padrão da linguagem Erlang:
  - erlang: manipulação de tuplas, concorrencia
  - lists: manipulação de listas
  - io: biblioteca entrada e saida
  - file: manipulação de arquivo
  - string: manipula cadeia de caracteres

Exemplos de funcionalidades das bibliotecas:

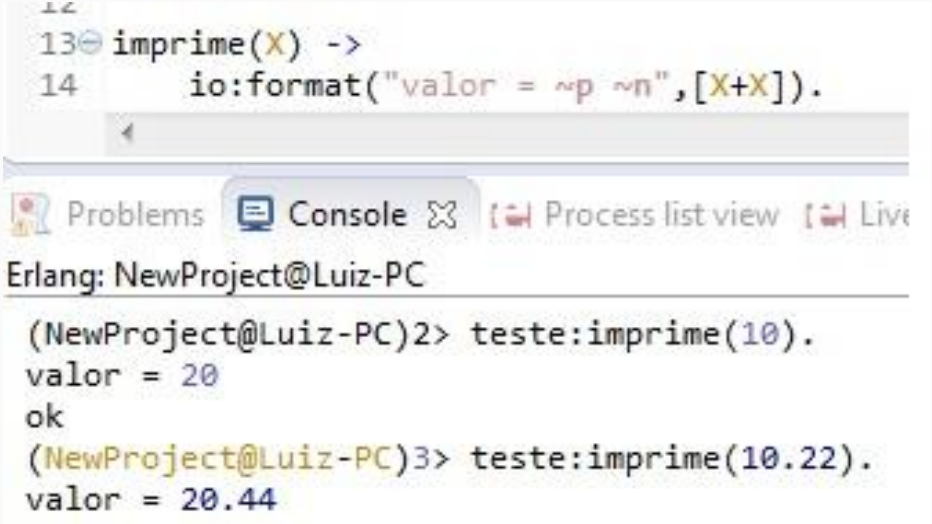
```
lists:keysearch(Key, N, TupleList),
lists:append(List1, List2),
lists:reverse(List1),
erlang:list_to_tuple(List),
{ok, ArquivoLido} = file:read_file(NomeDoArquivo),
ListaDeNomes = string:tokens(erlang:binary_to_list(ArquivoLido), ";\r\n").
```

Agenda

- ✓ Introdução
- ✓ Amarrações
- ✓ Valores e tipos de dados
- ✓ Variáveis e constantes
- ✓ Expressões e comandos
- ✓ Modularização
- ➢ **Polimorfismo**
- ➢ Exceções
- ➢ Concorrência
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Sistema de tipo

- Linguagem não tipada: Não necessita de declaração do tipo de nenhuma varável.
- Verificação dinâmica de tipos:
  - Prós: Maior eficiência;
  - Contra: Mais responsabilidade para o programador.



Polimorfismo

- Não suporta conversão de tipos;
- Tipos de polimorfismo:
  - Coerção;
  - Inclusão;
  - Paramétrico;
  - Sobrecarga;

Polimorfismo

- Exemplo polimorfismo paramétrico:

```
13 imprime(X) ->
14     io:format("valor = ~p ~n",[X]).
```

Problems | Console | Process list view | Live Expressio

Erlang: NewProject@Luiz-PC

```
20
(NewProject@Luiz-PC)63> teste:imprime(10).
valor = 10
ok
(NewProject@Luiz-PC)64> teste:imprime(1.2).
valor = 1.2
ok
(NewProject@Luiz-PC)65> teste:imprime("Hello World").
valor = "Hello World"
ok
```

Polimorfismo

- Exemplo polimorfismo sobrecarga:

```
13⊖ soma(X) ->
14       X+1.
15
16⊖ soma(X,Y) ->
17       X + Y.
(NewProject@Luiz-PC)61> teste:soma(10).
11
(NewProject@Luiz-PC)62> teste:soma(10,10).
20
```

Agenda

- ✓ Introdução
- ✓ Amarrações
- ✓ Valores e tipos de dados
- ✓ Variáveis e constantes
- ✓ Expressões e comandos
- ✓ Modularização
- ✓ Polimorfismo
- ➢ **Exceções**
- ➢ Concorrência
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Exceções

- Em Erlang as exceções podem ser geradas a partir de qualquer local do programa;
- Usa-se o metodo *try catch* para o tratamento de exceções.

Exceções

```
83 main1(NomeDoArquivo) ->
84
85 try {ok, Lista} = file:read_file(NomeDoArquivo) of
86     Val -> {normal, Val}
87     catch
88     exit:_ -> "Exit Exceptio: Informar valor correto";
89     throw:_ -> "Throw Exceptio: Informar valor correto";
90     error:_ -> "Exception error: Informar valor correto";
91         _:_ -> "Unknown exception "
92 end.
```

Problems | Console | Process list view | Live Expressions

Erlang: NewProject@Luiz-PC

```
(NewProject@Luiz-PC)12> teste:main1("a.txt").
"Exception error: Informar valor correto"
(NewProject@Luiz-PC)13> teste:main1("clientes.txt").
{normal,{ok,<<"Luiz Carlos;998495400\r\nJoao Paulo;123456">>}}
```

Agenda

- ✓ Introdução
- ✓ Amarrações
- ✓ Valores e tipos de dados
- ✓ Variáveis e constantes
- ✓ Expressões e comandos
- ✓ Modularização
- ✓ Polimorfismo
- ✓ Exceções
- ➢ **Concorrência**
- ➢ Avaliação da Linguagem
- ➢ Referências Bibliográficas

Concorência

- Em Erlang, como as “variavéis” são imutáveis, não existem efeitos colaterais ao se executar uma função, portanto podemos dizer que a “memória compartilhada” não sofre dos males das outras linguagens, já que não há estado que possa ser alterado;
- **Funções que não causam efeitos colaterais** permitem otimizações impressionantes por parte do interpretador/compilador e são uma ótima alternativa para fazer uso de todos os *cores* que temos disponibilizados (número que só cresce) evitando o desperdício;
- Em Erlang, a troca de contexto entre processos é cerca de duas vezes mais rápida que em C.
- Erlang tem suporte nativo para concorrência.

Concorência

- Ex: spawn(Module, Function, Args)
  - Retorna o pid de um novo processo e inicia a função aplicada do modulo:

```
1  -module(concorrencia).
2  -export([start/0,imprime/2]).
3
4
5
6  start() ->
7  spawn(concorrencia,imprime,[morto, 4]),
8  spawn(concorrencia,imprime,[vivo, 4]).
9
10 imprime(Palavra, 0) ->
11     done;
12 imprime(Palavra, X) ->
13     io:format("~p~n", [Palavra]),
14     imprime(Palavra, X - 1).
```

Problems | Console | Process list view

Erlang: NewProject@Luiz-PC

```
concorrencia:start().
morto
vivo
morto
vivo
<0.89.0>
morto
morto
vivo
vivo
(NewProject@Luiz-PC)4>
```

Agenda

- ✓ Introdução
- ✓ Amarrações
- ✓ Valores e tipos de dados
- ✓ Variáveis e constantes
- ✓ Expressões e comandos
- ✓ Modularização
- ✓ Polimorfismo
- ✓ Exceções
- ✓ Concorrência
- ➢ **Avaliação da Linguagem**
- ➢ Referências Bibliográficas

Critérios gerais

Critérios gerais	Erlang	C	Java
Aplicabilidade	Parcial	Sim	Parcial
Confiabilidade	Sim	Não	Sim
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Parcial
Portabilidade	Sim	Não	Sim
Método de Projeto	Funcional	Estruturado	OO
Evolutibilidade	Sim	Não	Sim
Reusabilidade	Sim	Parcial	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Critérios específicos

Critérios específicos	Erlang	C	Java
Escopo	sim	Sim	Sim
Expressões e comandos	sim	Sim	Sim
Tipos primitivos e compostos	sim	Sim	Sim
Persistência de dados	Biblioteca de funções (mnesia)	Biblioteca de funções	JDBC, biblioteca de classes, serialização
Passagem de parâmetros	Por valor	Lista variável e por valor	Lista variável, por valor e por cópia de referência

Critérios específicos

Critérios específicos	Erlang	C	Java
Encapsulamento e proteção	Não	Parcial	Sim
Sistema de tipos	Não	Não	Sim
Verificação de tipos	Dinâmica	Estática	Estática/Dinâmica
Polimorfismo	Sobrecarga e paramétrica	Coerção e sobrecarga	todos
Exceções	Sim	Não	Sim
Concorrência	Sim	Não	Sim
Gerenciamento de memória	Sim	Programador	Sistema

Conclusão

- A sintaxe é diferente das linguagens populares, dificultando o aprendizado;
- Aconselhavel usa-la apenas para programação concorrente e sistemas distribuidos;
- A linguagem Erlang é eficiênte e confiável;
- Possui suporte nativo para programação concorrente;
- O codigo pode ser atualizado sem necessidade de interromper sistema;
- Confiavel e eficiente;
- Pode ser integrado com Java, C/C++.

Agenda

- ✔ Introdução
- ✔ Amarrações
- ✔ Valores e tipos de dados
- ✔ Variáveis e constantes
- ✔ Expressões e comandos
- ✔ Modularização
- ✔ Polimorfismo
- ✔ Exceções
- ✔ Concorrência
- ✔ Avaliação da Linguagem
- ➢ **Referências Bibliográficas**

Referências

http://pt-br.grupo10lp.wikia.com/wiki/Erlang

www.erlang.com

http://elemarjr.net/2011/08/09/conceitos-fundamentais-de-erlang-parte-2-de-3/

http://pt-br.grupo10lp.wikia.com/wiki/Erlang

http://pt.slideshare.net/adorepump/a-linguagem-de-programao-erlang-presentation

Perguntas ???