What’s Erlang?

Erlang is a concurrent functional programming language, designed for writing concurrent programs that “run forever.”
History

- Developed in Ericsson Computer Science Laboratory (1986)
History

“Solve Ericsson’s software problem.”

SPOTS project

It was an initiative to find better ways to write telecom software under some requirements like...

- Very large number of concurrent activities
- Real time requirements
- Fault tolerance

---

Make the phones ring

To do so, they wrote POTS\textsuperscript{1} in almost every programming language available and some of the conclusions that came out were:

- “Small languages were thought desirable”\textsuperscript{2}
- “Functional programming was liked”
- “Logic programming was best in terms of elegance”
- Concurrency was essential

\textsuperscript{1}Plain Ordinary Telephone Service.
History

- Developed in Ericsson Computer Science Laboratory (1986)
- It was first implemented in Prolog
Joe Armstrong was working in his “POTS” smalltalk implementation alongside developing a graphical notation for defining his programs.

A certain day he showed his work to a colleague only to find him saying “but that’s a Prolog program.”

Joe quickly adapted his work to Prolog and threw away his smalltalk stuff.

---

History

✓ Developed in Ericsson Computer Science Laboratory (1986)
✓ It was first implemented in Prolog
■ Open Source Erlang was released in 1998
After Ericson banned the project in 1998, the development team obtained the approval to release Erlang at the end of the year.

Most of the Erlang development team left Ericsson, and started a new company called "Blue tail" for product development using Erlang.
History

✓ Developed in Ericsson Computer Science Laboratory (1986)
✓ It was first implemented in Prolog
✓ Open Source Erlang was released in 1998
Main features

- Concurrency
- Fault tolerance
- Garbage collection
- Functional
Suitable for...

- Large software for server use
- Higher-level protocol implementation
- Soft real-time systems
- Banking, e-commerce, computer telephony and instant messaging systems are also a good choice
Non suitable for...

- Number crunching
- Low level drivers
- Image processing
### Basic data types

**Numbers**

```plaintext
1> 1 + 2.0.
3.0

2> 2/3.
0.6666666666666666

3> 2 div 3.
0

4> 2*3+1.
7

5> 2*(3+1).
8
```
Basic data types

Variables

1> Var.
* 1: variable 'Var' is unbound
2> Var = 14.
   14
3> Var.
   14
4> Var + 1.
   15
5> Var = 34.
   ** exception error: no match of
   right hand side value 34
Basic data types

Atoms

1> am_an_atom.
am_an_atom
2> is_atom(am).
   true
3> Var=atom_1.
atom_1
4> is_atom(Var_1).
   * 1: variable ’Var_1’ is unbound
5> is_atom(Var).
   true
Basic data types

Boolean Algebra & Comparison operators

1> true and false.
false

2> true or
2> (false and true).
true

3> true or
3> false and true.
true

4> false and true or true.
true

5> 49 == 49.0.
true

6> 49 =:= 49.0.
false

7> 49 =/= 49.1.
true

8> 49 /= 49.0.
false
Basic data types

Tuples

1> { 15 , the_atom }.
{15, the_atom}

2> X = { 15 , the_atom }.
{15, the_atom}

3> { A, _ } = X.
{15, the_atom}

4> A.
15
Basic data types

Lists

1> [1, this, is, madness, 2, 3, 4].
   [1, this, is, madness, 2, 3, 4]
2> [X|Y] = [1, 3, 4, 5, 6, 7].
   [1, 3, 4, 5, 6, 7]
3> X.
   1
4> Y.
   [3, 4, 5, 6, 7]
Basic data types

Bit Syntax

0> \langle\langle \text{Sign}:1, \text{Exp}:8, \text{Man}:23\rangle\rangle = \langle\langle 255, 255, 255, 0\rangle\rangle.

1> \langle\langle 255, 255, 255, 0\rangle\rangle

1> Sign.

1

2> Man.

8388352

3> Exp.

255
Functions

Pattern matching

\[
\begin{align*}
\text{fibonacci}(0) & \rightarrow 0; \\
\text{fibonacci}(1) & \rightarrow 1; \\
\text{fibonacci}(X) & \rightarrow \text{fibonacci}(X-1) + \text{fibonacci}(X-2).
\end{align*}
\]
Functions

Pattern matching (Lists)

\[
\begin{align*}
\text{head}([X|\_]) & \rightarrow X \\
\text{length}([]) & \rightarrow 0 \\
\text{length}([\_|Y]) & \rightarrow 1 + \text{length}(Y).
\end{align*}
\]
what_is(X) when is_atom(X) ->
    "is an atom";
what_is(X) when is_list(X) ->
    "is a list";
what_is(X) when is_integer(X) ->
    "is a number".
Anonymous functions

fun(Args1) ->
    Expression1, Exp2, ..., ExpN;
(Args2) ->
    Expression1, Exp2, ..., ExpN;
(Args3) ->
    Expression1, Exp2, ..., ExpN
end
Other features

- Call by value
- Strong typing
- Dynamic typing
- High order functions
Concurrency

- “Is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other”

- “For many Erlangers, concurrency refers to the idea of having many actors running independently, but not necessarily all at the same time”

---

“Erlang uses the actor model, and each actor is a separate process in the virtual machine. In a nutshell, if you were an actor in Erlang’s world, you would be a lonely person, sitting in a dark room with no window, waiting by your mailbox to get a message.”

“Erlang’s actor model can be imagined as a world where everyone is sitting alone in their own room and can perform a few distinct tasks. Everyone communicates strictly by writing letters and that’s it.”

---

go() ->
    Pid2 = spawn(echo, loop, []),
    Pid2 ! {self(), hello},
    receive
        Pid2, Msg ->
            io:format("P1 ~w~n", [Msg])
        end,
        Pid2 ! stop.
loop() ->
    receive
        {From, Msg} ->
            From ! self(), Msg,
            loop();
        stop ->
            true
    end.
Use of Prolog for developing a new programming language\textsuperscript{1}

A practical subtyping system for Erlang \textsuperscript{2}

Practical Type Inference Based on Success Typings \textsuperscript{3}

HiPe

\textsuperscript{1}J. L. Armstrong, S. Virding, and M. C. Williams. Use of Prolog for developing a new programming language, 1992.
