Erlang: An Introduction

Lee Avital

March 31, 2014

Simple Syntax

Three List Length Functions

```erlang
list_length( [] ) -> 0;
list_length( [_H|T] ) -> 1 + list_length( T ).
```

```erlang
list_length2( L ) ->
  case L of
    [] -> 0;
    [_|T] -> 1 + list_length2( T )
  end.
```

```erlang
list_length3( L ) -> list_length3( L, 0 ).
list_length3( [], N ) -> N;
list_length3( [_|T], N ) -> list_length3(T, N + 1).
```

Powerful Pattern Matching

```
read_it( F ) when is_list( F ) ->
  {ok, D} = file:read_file( F ),
  io:format( "~p", [D] ).
```

```
read_it2( F ) when is_list( F ) ->
  case file:read_file( F ) of
    {ok, D} ->
      io:format( "~p", [D] );
    {error, Err} ->
      io:format( "error: ~p", [Err] )
    end.
```

Concurrency Primitives

```
ping( ) ->
  receive
    {ping, From} ->
      io:format( "ping ~n" ),
      timer:sleep( 500 ),
      From ! {pong, self()}
  end.
pong( ) ->
  timer:sleep( 500 ),
  Ping = spawn( ?MODULE, ping, [] ),
  Ping ! {ping, self()},
  receive
    {pong, _} ->
      io:format( "pong~n" )
  end.
ping_pong() -> pong().

ping_pong_async() -> spawn(?MODULE, pong, [] ).
```

Distributed Erlang

```
$ erl -sname larry
(larry) 1> L = fun
  receive {P, Msg} ->
    P ! Msg
  end
end.
(larry 2> spanw( L ).
```