Erlang
Message Passing
Concurrency, For The Win

Toby DiPasquale
Commerce360, Inc.
Presented to Philly on Rails
September 12, 2007
Er-what?

- Language/runtime created at Ericsson
- Designed for scalable, long-lived systems
- Not (explicitly) object oriented
Another f***** language?

- Pattern matching
- Tail call optimization
- Message-passing concurrency
- Distributed programming
- Hot code update
Runway Models

- Meebo
- SlideAware
- RabbitMQ
- Jabber.org
- OpenPoker
Sequential
Syntax

- Variables can only be assigned once
- Variables start with uppercase letter
- Last evaluation is return value of function
“You keep using this symbol. I do not think it means what you think it means.”

Not a mutation (assignment) in Erlang

\[ \text{LHS} = \text{RHS} \]

evaluate RHS and match against LHS pattern

much more leeway to make it true
She's got the look

-module(math_o_matics).
-export([square/1]).

define(square, X) ->
    X * X.

define(cube, X) ->
square(X) * X.
Atoms

- Self-indicating identifiers
- Start with lowercase letter
- Can also be quoted with single quotes

```plaintext
atom
this_is_an_atom
'I am also an atom'
```
Tuples

- Fixed length containers
- Often prepended with an identifier atom
- Decompose with pattern matching

```
Car = {car,
    {honda, civic},
    {horsepower, 100}}.
{car, Type, Power} = Car.
```
Lists

- Variable length containers
- Use [H|T] syntax to get head and tail of list

List = [1, 2, 3, four, 5.0]

[Head|Tail] = List

[H1,H2|T2] = List
Strings

- Sort of like in C
- Strings are just lists of integers
- Must use double quotes

Meeting = “PLUG”.

Meeting2 = [80, 76, 85, 71].
Arity

- Use functions with same name and different arity* as auxiliary functions

-module(math_o_matics).
-export([[sum/1]]).

sum(L) -> sum(L, 0).
sum([], N) -> N;
sum([H|T], N) -> sum(T, H+N).

* ‘Arity’ refers to the number of input parameters a function takes
Modules

- Logically associated code block
- Use colon (:) to use intermodule code
- Use -import to avoid prefixing

io:format("Using the module io~n").
The “fun” in functional

- Anonymous functions
- Used for higher-order programming

\[
\text{Square} = \text{fun}(X) \rightarrow X \times X \text{ end.}
\]

\[
\text{Cube} = \text{fun}(X) \rightarrow \text{Square}(X) \times X \text{ end.}
\]
List Comprehensions

- Takes an expression and a set of qualifiers and returns another list (like Python’s)
- Looks like: 

\[ [X \mid Q1, Q2, \ldots, Qn] \]

```prolog
qsort([]) -> []; 
qsort([Pivot|T]) -> 
  qsort([X \mid X <- T, X < Pivot]) 
++ [Pivot] ++ 
qsort([X \mid X <- T, X >= Pivot]).
```
Guards

- Simple tests against a pattern matching
- Makes code more concise and readable

\[
\begin{align*}
\max(X, Y) & \text{ when } X > Y \rightarrow X; \\
\max(X, Y) & \rightarrow Y.
\end{align*}
\]
Biting the bits

- Syntax for extracting/packing bits
- Very handy for binary protocols (IPv4, MPEG, etc)

```c
<<?IP_VERSION:4,
   HLen:4, SrvcType:8, TotLen:16,
   ID:16, Flgs:3, FragOff:13,
   TTL:8, Proto:8, HdrChkSum:16,
   SrcIP:32, DestIP:32, RestDgram/binary>>
```
Concurrency
Shared Memory

Image credit: http://www.ja.org/nested/berrienandcass/kelly-vault.jpg
Message Passing

Processes

- Basic unit of concurrency
- Managed by runtime, not OS (i.e. cooperative)
- Use `spawn/0, !/1` (a.k.a. send) and `receive/1` BIF’s*
- Asynchronous send, synchronous receive

* BIF means “Built-in Function”
Concurrency Template

-module(template).
-compile(export_all).

start() ->
    spawn(fun() -> loop([]) end).

rpc(Pid, Query) ->
    Pid ! {self(), Query},
    receive
        {Pid, Reply} ->
            Reply
    end.

loop(X) ->
    receive
        Any ->
            io:format("Received:~p~n", [Any]),
            loop(X)
    end.

Courtesy of Joe Armstrong in Programming Erlang, First Edition
Errors

- Linking processes defines error chain
- When a process dies, linked processes are sent an exit signal
- Use `spawn_link/1` to spawn linked processes
Distributing Erlang

Erlang has built-in support for distributed operation

Two modes:

- Distributed Erlang (easier, less secure)
- Socket-based distribution (more secure)
Distributing Erlang (2)

- Two libraries for higher-level Distributed Erlang:
  - rpc - RPC services
  - global - naming, locking, maintenance
- Cookie based security model
ets and dets

- Erlang Term Storage
- Dictionary for mad loads of Erlang data
- ets tables are RAM-based (transient)
- dets (disk ets) tables are persisted to disk
Mnesia

- Real-time, distributed database that comes with Erlang
- Query language looks like a lot like SQL/list comprehensions
- Built-in visualization tools
OTP

- Open Telecom Platform
- Not just for telco ;-) 
- HTTP server, FTP server, CORBA ORB, ASN.1, SNMP, etc
- Designed around encapsulated "behaviors"
OTP Behaviours

- Standard application framework
- Behavior hosts non-functional aspects
- You supply functional aspects in “callbacks”
- Similar in concept to J2EE Container
- Check out gen_server
Hot Process-on-Process action

- **Yaws**
  - Super scalable Web server/platform

- **ejabberd**
  - Super scalable **XMPP** (Jabber) server

- **RabbitMQ**
  - Super scalable message broker
RTFM

- Programming Erlang (PDF and dead tree versions; great book)
- Concurrent Programming with Erlang (older; first half available online at no cost)
- Erlang Website
- Trapexit forums
- erlang-questions mailing list
Huh huh huh huh huh... you said ‘Erlang’

Slides: http://cbcg.net/talks/erlang.pdf

Me: toby@cbcg.net