```
1> clicker_test_SUITE:next().
```

```
1> clicker_test_SUITE:next().
ok
2>
```

Introduction to Erlang

Iñaki Garay | Erlounge BA 2014

inaka

Fasten your seatbelts

General language traits

Erlang is about building reliable systems.

Runtime designed for the development of distributed, soft real-time, fault-tolerant, continuous use systems.

Now useful for scalability.

General language traits

inaka
Erlounge BA | 2014

- simple
- **functional**
- strong, **dynamic** typing (late binding)
- immutability
- garbage-collected
- actors as concurrency model

General language traits

inaka
Erlounge BA | 2014

- simple
- **functional**
- strong, **dynamic** typing (late binding)
- immutability
- garbage-collected
- actors as concurrency model

General language traits

inaka
Erlounge BA | 2014

- simple
- **functional**
- strong, **dynamic** typing (late binding)
- immutability
- garbage-collected
- actors as concurrency model

Types & Values

inaka
Erlounge BA | 2014

```
int 2, 1099511627776
float 1.2
atom ok, error, '$POST'
fun fun(X) -> X * X end

ref #Ref<0.0.0.47>
pid <0.36.0>
port #Port<0.589>
```

```
tuple {1,2,3,a,3.14}
list [], [a,b,c], [{a,b}, 123]

string "400 (Bad Request)"

binary <<1,1,2,3,5,8,13,21>>,
<<"{\"id\":42}">>
```

Types & Values

inaka
Erlounge BA | 2014

Records:

```
-record(user, {id, posts}).

#user{id=42, posts=31}      =      {user,31,42}
```

Code organization

inaka
Erlounge BA | 2014

```
to_binary({date, {Y, M, D}}) ->
  Year  = integer_to_binary(Y),
  Month = integer_to_binary(M),
  Day   = integer_to_binary(D),
  <<Day/binary, "/", Month/binary, "/", Year/binary>>.

1> to_binary({date, {2014, 4, 28}}).
<<"28/4/2014">>
```

Code organization

inaka
Erlounge BA | 2014

```
to_binary({date, {Y, M, D}}) ->
  Year  = integer_to_binary(Y),
  Month = integer_to_binary(M),
  Day   = integer_to_binary(D),
  <<Day/binary, "/", Month/binary, "/", Year/binary>>;
to_binary({datetime, {Y, M, D}, {H, N, S}}) ->
  ...
```

Code organization

inaka
Erlounge BA | 2014

in web_handler.erl:

```
-module(web_handler).
-export([process_json/3]).

process_json(post, Req, State) ->
  ...;
process_json(put, Req, State) ->
  ....
```

elsewhere:

```
web_handler:process_json(...)


Module   = web_handler,
Function = process_json,
Module:Function(...),
```

Variables

Single assignment (immutability).

```
1> Variable = value.
2> A = 1.
1
3> A = 2.
** exception error: no match of right hand side
value 2
```

Pattern Matching

inaka
Erlounge BA | 2014

```
1> {A, B} = {answer, 42}.
{answer,42}
2> A.
answer
3> {A, B} = {q, 42}.
** exception error: no match of right hand side
value {q, 42}.
```

Pattern Matching

inaka
Erlounge BA | 2014

```
valid_time({ {Y,M,D}, Time = {H,N,S}}) ->
   {valid, Time};
valid_time(_) ->
   error.

<<SourcePort:16, DestinationPort:16,  AckNumber:32,
  DataOffset:4, Reserved:4, Flags:8, WindowSize:16,
  CheckSum:16, UrgentPointer:16,
  Payload/binary>> = RawTCPHeader.
```

Control flow

inaka
Erlounge BA | 2014

Control flow is done with functions, recursion, case and receive expressions and pattern matching.

```
function_head(Argument) ->
    case process_request(InitialState) of
      {ok, Result} ->
        {Result, Req, State};
      {error, {notfound, EntityId}} ->
        return(404, Req, State)
    end.
```

Control flow

inaka
Erlounge BA | 2014

```
member(X, [])        -> false.
member(X, [X | _]) -> true;
member(X, [_ | Y]) -> member(X, Y);

1> [{X, X} || X <- [1,2,3], X /= 2].
[{1,1},{3,3}]
```

Concurrency

inaka
Erlounge BA | 2014

```
1> F = fun() -> 2 + 2 end.
#Fun<erl_eval.20.67289768>
2> Pid = spawn(F).
<0.44.0>
```

Concurrency

inaka
Erlounge BA | 2014

```
1> Pid = spawn(web_handler, process_json, []).
<0.44.0>
```

Concurrency

inaka
Erlounge BA | 2014

```
3> Me = self().
<0.33.0>
4> Me ! message.
message
5> receive message -> got_it end.
got_it
```

Concurrency

inaka
Erlounge BA | 2014

```
receive
    Pattern1 ->
        Body1;
    ...;
    PatternN ->
        BodyN
after
   Timeout1 ->
end
```

Concurrency

inaka
Erlounge BA | 2014

```
loop() ->
    receive
        {type1, Message} ->
            process1(Message);
        {type2, Message} ->
            process2(Message)
    after
        5000 -> throw(timeout)
    end,
    loop().
```

Missing

Many topics not mentioned

- ➔ linking, monitoring, error handling
- ➔ hot code loading
- ➔ distribution
- ➔ OTP and standard library
- ➔ more behaviors/programming patterns
- ➔ all the “buts” and exceptions
- ➔ design philosophy

Online References

http://www.erlang.org/doc/index.html

http://learnyousomeerlang.com/

Thank you!

inaka
Erlounge BA | 2014

igaray

@iraunkortasuna

```
broadcast(Subscribers, Message) ->
  [ Pid ! {broadcast, Message}
  || #subscription{pid = Pid} <- Subscribers
  ].
```