Programmeertalen: Erlang

A distributed functional programming language

Jeroen Koops, Jurre Brandsen, Bryan Westerveld, Ana Oprescu

Universiteit van Amsterdam

3 maart 2020

Programmeertalen: where are we now?

	Programming Language	**Lecture Concepts**[1]	**Lecture Best Practices**[2]	
Week 1	Bash	Tue 4/2	Thu 6/2	
Week 2	Haskell	Tue 11/2	Thu 13/2	
Week 3	Prolog	Tue 18/2	Thu 20/2	
Week 4	Python	Tue 25/2	Thu 27/2	
Week 5	**Erlang**	Tue 3/3	**Thu 5/3**	
Week 6	Go	Tue 10/3	Thu 12/3	
Week 7	C++	Tue 17/3	Thu 19/3	
Week 8	Exam	Tue 24/3	09:00-12:00	Sporthal 2

[1] 11:00-13:00, CWI Turingzaal
[2] 13:00-15:00, see DataNose

Today's menu

Callbacks

find.erl

```
find_less_than(L, Limit) -> find_less_than(L, Limit, []).

find_less_than([], _Limit, Acc) -> Acc;
find_less_than([H|T], Limit, Acc) when H < Limit ->
    find_less_than(T, Limit, [H|Acc]);
find_less_than([_|T], _Limit, Acc) ->
    find_less_than(T, Limit, Acc).
```

Callbacks

find.erl

```
find_less_than(L, Limit) -> find_less_than(L, Limit, []).

find_less_than([], _Limit, Acc) -> Acc;
find_less_than([H|T], Limit, Acc) when H < Limit ->
    find_less_than(T, Limit, [H|Acc]);
find_less_than([_|T], _Limit, Acc) ->
    find_less_than(T, Limit, Acc).
```

find.erl

```
find_greater_than(L, Limit) -> find_greater_than(L, Limit, []).

find_greater_than([], _Limit, Acc) -> Acc;
find_greater_than([H|T], Limit, Acc) when H > Limit ->
    find_greater_than(T, Limit, [H|Acc]);
find_greater_than([_|T], _Limit, Acc) ->
    find_greater_than(T, Limit, Acc).
```

Callbacks

find.erl

```
find_between(L, Min, Max) -> ....
```

Callbacks

find.erl

```
find_between(L, Min, Max) -> ....
```

There must be a better way!

Callbacks

find.erl

```
find(L, Fun) -> find(L, Fun, []).

find([], _Fun, Acc) -> Acc;
find([H|T], Fun, Acc) ->
    case Fun(H) of
        true -> find(T, Fun, [H|Acc]);
        false -> find(T, Fun, Acc)
    end.
```

Callbacks

find.erl

```
find(L, Fun) -> find(L, Fun, []).

find([], _Fun, Acc) -> Acc;
find([H|T], Fun, Acc) ->
    case Fun(H) of
        true -> find(T, Fun, [H|Acc]);
        false -> find(T, Fun, Acc)
    end.
```

```
1> F = fun(N) when N > 2 -> true;
          (_)            -> false
     end.

2> find([2, 3, 4, 5], F).
[3,4,5]
```

Callbacks

- in the context of generic encapsulated behaviour, such as libraries, callbacks are usually user-defined behaviour
- to allow customization of the generic behaviour
- examples are signal handlers in operating systems, or event handlers in a UI library

Callbacks

- in Erlang callbacks are used extensively to make generic behaviours (such as gen_servers) perform application specific logic.
- You already saw this in the Tic Tac Toe exercise.

Synchronous versus asynchronous

- synchronous: the sender waits for a reply
- asynchronous: the sender "informs" without waiting for a reply
- When an asynchronous Erlang function does want to send a reply to the caller, it can send a message
- Or the caller supplies a callback function, which will be called when a result is available

Example: bankaccount

- Start with a begin balance
- Function to get current balance { `ok, 49.50` }
- Function to withdraw amount { `ok, 39.50` } or { `error, insufficient_funds` }

Example: bankaccount (first attempt)

account.erl

```
-module(account).

-behaviour(gen_server).

-export([ start/1, balance/1, withdraw/2 ]).
-export([ init/1, handle_call/3, handle_cast/2 ]).

start(InitialBalance) -> gen_server:start(account, InitialBalance, []).

balance(Pid) -> gen_server:call(Pid, balance).
```

Example: bankaccount (first attempt)

account.erl

```
withdraw(Pid, Amount) ->
    case balance(Pid) of
        Balance when Balance < Amount ->
            { error, insufficient_funds };
        _ ->
            gen_server:cast(Pid, { set_balance, Balance - Amount }),
            { ok, Balance - Amount }
    end.
```

Example: bankaccount (first attempt)

account.erl

```
init(InitialBalance) -> { ok, InitialBalance }.

handle_call(balance, _From, Balance) -> { reply, { ok, Balance }}.

handle_cast({ set_balance, NewBalance }, _Balance) -> { noreply, NewBalance }.
```

Example: bankaccount (second attempt)

account.erl

```
-module(account).

-behaviour(gen_server).

-export([ start/1, balance/1, withdraw/2 ]).
-export([ init/1, handle_call/3, handle_cast/2 ]).

start(InitialBalance) -> gen_server:start(account, InitialBalance, []).

balance(Pid) -> gen_server:call(Pid, balance).

withdraw(Pid, Amount) -> gen_server:call(Pid, { widthdraw, Amount }).
```

Example: bankaccount (second attempt)

account.erl

```
init(InitialBalance) -> { ok, InitialBalance }.

handle_call(balance, _From, Balance) -> { reply, { ok, Balance }};

handle_call({ withdraw, Amount }, _From, Balance) when Amount > Balance ->
    { reply, { error, insufficient_funds }, Balance };

handle_call({ withdraw, Amount }, _From, Balance) ->
    NewBalance = Balance - Amount,
    { reply, { ok, NewBalance }, NewBalance }.

handle_cast(_, Balance) -> { noreply, Balance }.
```

Asynchronous response from gen_server

async.erl

```
handle_call(..., _From, State) -> { reply, ..., State };

handle_call(..., From, State) -> { noreply, store_caller(State) };

....

handle_info(..., State) ->
    From = get_caller(State),
    gen_server:reply(From, ...).
```

Macros

macros.erl

```
-define(NR_ROWS, 3).
-define(LOG(X), io:format("Alert: ~s~n", [ X ])).

lists:seq(1, ?NR_ROWS).

?LOG("Max temperature exceeded").
```

Idiomatic Erlang

Q&A - Erlang in general

Important concepts visited in Erlang by Pinky and The Brain

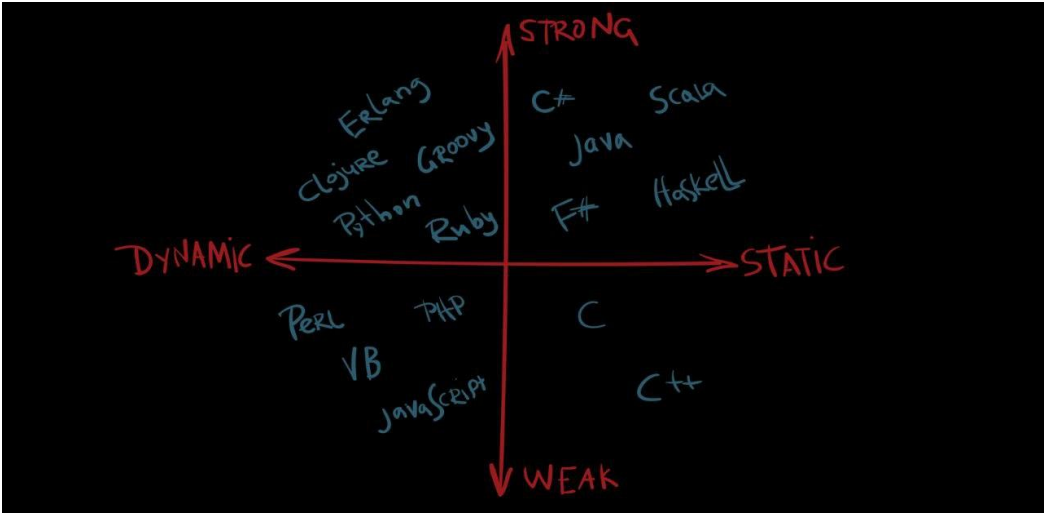
- Compiling versus interpreting
- Tail-recursion
- Higher-order functions
- Anonymous (lambda) functions
- Lazy evaluation
  - Generators
- Types of type systems
- Types of state

Important concepts visited in Erlang by Pinky and The Brain

First, a Kahoot to see what you already know of Erlang!

Please visit kahoot.it and fill in the PIN-code.

Types of type systems



Refactoring your code (1)

tictactoe.erl

```
get_character(V) ->
    if
        V =:= 0 ->
            " ";
        V =:= 1 ->
            "O";
        V =:= 2 ->
            "X"
    end.
```

- You can start with a simple working solution, but shouldn't stop there.

Refactoring your code (2)

tictactoe.erl

```
get_character(V) ->
    case V of
        0 -> " ";
        1 -> "O";
        2 -> "X"
    end.
```

Refactoring your code (3)

tictactoe.erl

```
get_character(0) ->
    " ";
get_character(1) ->
    "O";
get_character(2) ->
    "X".
```

- Keep this example in mind for the team assignment.

Refactoring your code (4)

tictactoe.erl

```
check_pos_open([], X, Y) ->
    true;
check_pos_open([ { PosX, PosY, _ } | Board ], X, Y) ->
    if
        PosX == X andalso PosY == Y ->
            false;
        true ->
            check_pos_open(Board, X, Y)
    end.
```

Refactoring your code (5)

tictactoe.erl

```
check_pos_open([], X, Y) ->
    true;
check_pos_open([ { X, Y, _ } | Board ], X, Y) ->
    false;
check_pos_open([ H | Board ]), X, Y) ->
    check_pos_open(Board, X, Y).
```

Refactoring your code (6)

tictactoe.erl

```
check_pos_open(Board, X, Y) ->
    lists:any(lists:map(fun(V) -> lists:member({ X, Y, V } end, Board)), [1, 2]
```

- You can start with a simple working solution, but shouldn't stop there.

Team Assignment

- Who's not familiar with the game 'Kamertje verhuren'?
  - The first paragraph of the team assignment displays the rules.
  - Let us demonstrate the coordinate system, since a few students already asked us about this!

Q&A - Erlang Team Assignment

- coordinate system → pay attention to the inverted Y-axis. For more details, see the recording of the lecture[3] from 1:19:00.
- random function → use rand, not random
- the AI should always complete a room if it is available, otherwise pick a rand(om) wall.
- generating players is asynchronous (so, no guaranteed order).

[3] `https://webcolleges.uva.nl/Mediasite/Play/250739e37c27440d82c4283ac6d7a7021d`