Programmeertalen: Erlang

Een functionele gedistribueerde programmeertaal

Robin de Vries

Universiteit van Amsterdam

5 maart 2018

Programmeertalen: waar zijn we nu?

	Taal	**Hoorcollege**			
Week 1	Haskell	ma 5/2	9:00-11:00	C0.05	Robin
Week 2	Bash	ma 12/2	9:00-11:00	C0.05	Bas
Week 3	Prolog	ma 19/2	9:00-11:00	C0.05	Robin
Week 4	Python	ma 26/2	9:00-11:00	C0.05	Bas
Week 5	**Erlang**	**ma 5/3**	**9:00-11:00**	**C0.05**	**Robin**
Week 6	Go	ma 12/3	9:00-11:00	C0.05	Robin
Week 7	C++	ma 19/3	9:00-11:00	C0.110	Bas
Week 8	Tentamen	wo 28/3	13:00-16:00	USC Sporthal 1	

Hoe ging Python?

Inhoud

Erlang

- Ontwikkeld door Ericsson
- Verschenen in 1986
- Gedistribueerd, functioneel
- Dynamisch en sterk getypeerd
- Fout tolerant
  - Let it crash!
- Bytecode draait op een VM

Karakteristieken

- Functionele taal
- Prolog-achtige syntax
- Ondersteuning voor concurrency

Erlang leren[1]

- Learn You some Erlang for Great Good! `http://learnyousomeerlang.com/content`
- An Erlang Course `http://www.erlang.org/course/course.html`
- Erlang/OTP documentation `http://www.erlang.org/doc/`

[1] Enkele slides en voorbeelden zijn afkomstig van dit materiaal.

Files, modules, comments and Hello World!

hello.erl

```
-module(hello).
-export([hello_world/0]).

% Say hello!
hello_world() -> io:fwrite("Hello World!\n").
```

```
Eshell V6.4  (abort with ^G)
1> c(hello).
{ok,hello}
2> hello:hello_world().
Hello World!
ok
```

Datatypen

- Numbers
- Variables (single assignment, beginnen met een hoofdletter)
- Atoms (beginnen met een kleine letter)
- Tuples
- Lists

Aritmetiek

```
1> 3 + 20.
23
2> 5 rem 4.
1
3> 19 div 3.
6
4> false and true.
false
5> not false.
true
6> 3 / 2.
1.5
```

En nog meer zoals `or`, `xor`, `andalso` en `orelse`.

(On)gelijkheid

- Testen op (on)gelijkheid: `=:=` en `=/=`
- Testen op (on)gelijkheid met int/float conversie: `==` en `/=`

```
1> 1 >= 1.
true
2> 1 =< 1.
true
3> 8.0 =:= 8.
false
4> 8.0 == 8.
true
5> fout == fout.
true
```

Inhoud

Lijsten

Homogene verzameling van elementen. Homogeniteit niet afgedwongen door Erlang.

Lijsten

Homogene verzameling van elementen. Homogeniteit niet afgedwongen door Erlang.

```
1> X = [1,2,3,4,5,6].
[1,2,3,4,5,6]
2> [H|T] = X.
[1,2,3,4,5,6]
3> H.
1
4> T.
[2,3,4,5,6]
5> hd(X).
1
6> tl(X).
[2,3,4,5,6]
7> length(X).
6
```

Strings

```
1> [97,98,99].
"abc"
```

Strings

Strings zijn een lijst van integers.

En de interpreter kiest automatisch de representatie naar keuze.

```
2> [1|"hoooi"].
[1,104,111,111,111,105]
```

Lijsten

Met de ++ operator kun je twee lijsten aan elkaar vast plakken. De -- operator doet het omgekeerde. Beide operatoren zijn *right-associative*.

```
1> [1,2,3] ++ "abc" ++ "def".
[1,2,3,97,98,99,100,101,102]

2> [1,2,3] -- [1,2] -- [3].
[3]

3> [1,2,3] -- [1,2] -- [2].
[2,3]

4> ([1,2,3] -- [1,2]) -- [3].
[]
```

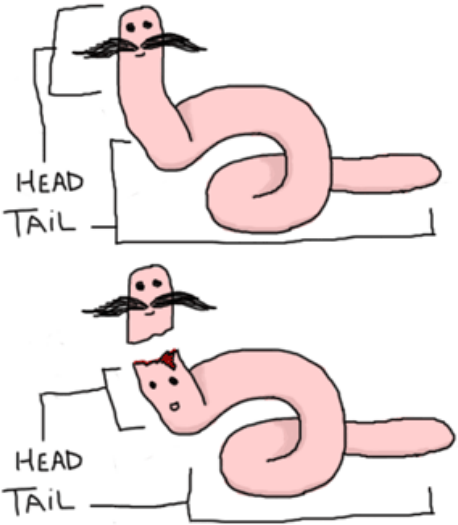
Lijsten

Vergelijkbaar met Prolog:

```
1> [1,2,3,4,5,6,7].
[1,2,3,4,5,6,7]

2> [1,2 | [3,4,5,6,7]].
[1,2,3,4,5,6,7]

3> [1,2,3 | [4,5,6,7]].
[1,2,3,4,5,6,7]
```



Meer hulpfuncties?

Raadpleeg de documentatie van de ingebouwde `lists`-module:
`http://www.erlang.org/doc/man/lists.html`

List comprehensions

Een krachtige manier om lijsten op te bouwen, net zoals we gezien hebben in Haskell.

```
1> L = lists:seq(1,20).
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

2> [ X || X <- L, X rem 2 =:= 0].
[2,4,6,8,10,12,14,16,18,20]

3> [ 2 * X || X <- L, X rem 2 =:= 1].
[2,6,10,14,18,22,26,30,34,38]
```

Pythagoras triplets?

List comprehensions

Een krachtige manier om lijsten op te bouwen, net zoals we gezien hebben in Haskell.

```
1> L = lists:seq(1,20).
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

2> [ X || X <- L, X rem 2 =:= 0].
[2,4,6,8,10,12,14,16,18,20]

3> [ 2 * X || X <- L, X rem 2 =:= 1].
[2,6,10,14,18,22,26,30,34,38]
```

Pythagoras triplets?

```
4> [ {X,Y,Z} || X <- L, Y <- L, Z <- L, X*X+Y*Y =:= Z*Z, X < Y].
[{3,4,5},{5,12,13},{6,8,10},{8,15,17},{9,12,15},{12,16,20}]
```

Oefening

Definieer een eigen versie van `lists:reverse()/1`.

```
1> lists:reverse("Hoooi").
"ioooH"
```

Oefening

Definieer een eigen versie van `lists:reverse()/1`.

```
1> lists:reverse("Hoooi").
"ioooH"
```

lecture.erl

```
reverse(X)        -> reverse(X, []).
reverse([], X)    -> X;
reverse([H|T], Y) -> reverse(T, [H|Y]).
```

```
2> lecture:reverse("Hoooi").
"ioooH"
```

Oefening

Definieer een eigen versie van `lists:member/2`.

```
1> lists:member(1,[1,2,3]).
true
2> lists:member(4,[1,2,3]).
false
```

Oefening

Definieer een eigen versie van `lists:member/2`.

```
1> lists:member(1,[1,2,3]).
true
2> lists:member(4,[1,2,3]).
false
```

lecture.erl

```
member(_, [])     -> false;
member(E, [E|_]) -> true;
member(E, [_|T]) -> member(E, T).
```

```
3> lecture:member(1,[1,2,3]).
true
4> lecture:member(4,[1,2,3]).
false
```

Inhoud

Pattern-matching en unbound-variables

lecture.erl

```
lucky(4) -> lucky;
lucky(6) -> doomed;
lucky(7) -> very_lucky;
lucky(_) -> not_so_lucky.
```

```
1> lecture:lucky(6).
doomed
2> lecture:lucky(7).
very_lucky
3> lecture:lucky(10).
not_so_lucky
```

Guards

Ouder dan 18?

```
old_enough(0)  -> false;
old_enough(1)  -> false;
old_enough(2)  -> false;
...
old_enough(16) -> false;
old_enough(17) -> false;
old_enough(_)  -> true.
```

Guards

Ouder dan 18?

```
old_enough(0)  -> false;
old_enough(1)  -> false;
old_enough(2)  -> false;
...
old_enough(16) -> false;
old_enough(17) -> false;
old_enough(_)  -> true.
```

Met guards:

```
old_enough(X) when X >= 18 -> true;
old_enough(_)              -> false.
```

Guards (2)

Ouder dan 18, niet ouder dan 104

```
right_age(X) when X >= 18, X =< 104 -> true;
right_age(_)                        -> false.
```

Let op het verschil tussen de , en de ;

```
wrong_age(X) when X < 18; X > 104 -> true;
wrong_age(_)                      -> false.
```

If-statements Erlang-style

```
answer_to_life(X) ->
    if
        X =:= 42  -> true;
        X =:= 666 -> false;
        true      -> false  % else-statement Erlang-style
    end.
```

Case

```
lucky_case(X) ->
    case X of
        4 -> lucky;
        6 -> doomed;
        7 -> very_lucky;
        _ -> not_so_lucky
    end.
```

Case met when

```
beach(Temperature) ->
  case Temperature of
    {celsius, N} when N >= 20, N =< 45     -> 'favorable';
    {kelvin, N} when N >= 293, N =< 318    -> 'scientifically favorable';
    {fahrenheit, N} when N >= 68, N =< 113 -> 'favorable in the US';
    _                                      -> 'avoid beach'
  end.
```

Invoer en uitvoer

lecture.erl

```
length_input() ->
   X = io:get_line("Geef een zin: "),
   L = length(X),
   io:format("Ingevoerde zin: ~s, lengte: ~w~n",[X,L]).
```

Informatie over alle functies in de IO-module: `http://erlang.org/doc/man/io.html`

Inhoud

Bestaande functies als argumenten

lecture.erl

```
one() -> 1.
two() -> 2.

add(X,Y) -> X() + Y().
```

Hoe roepen we add/2 aan zodat er 3 uitkomt?

Bestaande functies als argumenten

lecture.erl

```
one() -> 1.
two() -> 2.

add(X,Y) -> X() + Y().
```

Hoe roepen we add/2 aan zodat er 3 uitkomt?

```
1> lecture:add(fun lecture:one/0,fun lecture:two/0).
3
```

Anonieme functies

Syntax:

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

Voorbeeld:

```
1> lists:map(fun(A) -> A + 1 end, [1,2,3]).
[2,3,4]

2> lists:filter(fun(A) -> A > 10 end, lists:seq(1,20)).
[11,12,13,14,15,16,17,18,19,20]
```

Anonieme functies (2)

Ook Erlang kent foldl en foldr, maar de syntax is anders dan bij Haskell.

```
1> lists:foldl(fun(X,Y) -> X+2*Y end, 4, [1,2,3]).
43

2> lists:foldr(fun(X,Y) -> X+2*Y end, 4, [1,2,3]).
49
```

Inhoud

Processen spawnen

Met behulp van `spawn` kunnen we processen opstarten. `spawn` geeft als return waarde de Process Identifier (PID) van het nieuwe process.

```
1> F = fun() -> io:format("Hello process!~n") end.
#Fun<erl_eval.20.90072148>
2> F().
Hello process!
ok
3> spawn(F).
Hello process!
<0.43.0>
```

Processen spawnen (2)

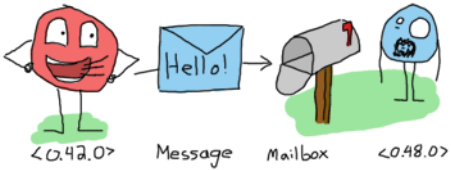
lecture.erl

```
hello_process(X) ->
    io:format("Hello ~w~n",[X]).
```

```
1> spawn(lecture, hello_process,[world]).
Hello world
```

Berichten versturen

Met de !-operator (bang symbol) kun je berichten naar processen sturen.




```
3> self() ! hallo.
hallo
4> self() ! wereld.
wereld
5> flush().
Shell got hallo
Shell got wereld
ok
```

Berichten ontvangen

lecture.erl

```
hi_process() ->
    receive
        {print, Message} -> io:format("Message: ~s~n",[Message]),
                            hi_process();
        stop             -> io:format("Goodbye!~n");
        _                -> io:format("What?~n"),
                            hi_process()
    end.
```

Berichten ontvangen

lecture.erl

```
hi_process() ->
    receive
        {print, Message} -> io:format("Message: ~s~n",[Message]),
                            hi_process();
        stop             -> io:format("Goodbye!~n");
        _                -> io:format("What?~n"),
                            hi_process()
    end.
```

```
1> P = spawn(lecture,hi_process,[]).
<0.111.0>
2> P ! {print, "Hello!"}.
Message: Hello!
{print,"Hello!"}
3> P ! {print, "World"}.
Message: World
```

Functioneel denken met processen

Hoe maken we in één keer tien processen aan?

Functioneel denken met processen

Hoe maken we in één keer tien processen aan?

lecture.erl

```
hi_process(N) ->
    receive
        {print, Message} -> io:format("~w: Message: ~s~n",[N,Message]),
                            hi_process(N);
        stop             -> io:format("Goodbye from ~w!~n",[N]);
        _                -> io:format("~w What?~n",[N]),
                            hi_process(N)
    end.
```

```
1> L = [ spawn(lecture, hi_process, [X]) || X <- lists:seq(1,10) ].
[<0.46.0>,<0.47.0>,<0.48.0>,<0.49.0>,<0.50.0>,<0.51.0>,
 <0.52.0>,<0.53.0>,<0.54.0>,<0.55.0>]
```

Functioneel denken met processen (2)

En hoe sturen we naar alle 10 processen een bericht?

Functioneel denken met processen (2)

En hoe sturen we naar alle 10 processen een bericht?

```
2> lists:foreach( fun(P) -> P ! {print, "Haai"} end, L).
1: Message: Haai
2: Message: Haai
3: Message: Haai
4: Message: Haai
5: Message: Haai
6: Message: Haai
7: Message: Haai
8: Message: Haai
9: Message: Haai
10: Message: Haai
ok
```

Er is geen enkele garantie dat de berichten in deze volgorde aankomen!

Inspiratie voor de opdracht: rotate game!

lecture.erl

```
rotate([H|T]) -> T ++ [H].

rotate_game(X) ->
    receive
        stop     -> io:format("~w game is over~n",[X]);
        {25, Ps} -> io:format("~w ends game~n",[X]),
                    lists:foreach( fun(P) -> P ! stop end, Ps);
        {N, Ps } -> io:format("~w increases ~w~n",[X, N]),
                    hd(Ps) ! {N + 1, rotate(Ps)},
                    rotate_game(X)
    end.
```

Inspiratie voor de opdracht: rotate game!

lecture.erl

```
rotate([H|T]) -> T ++ [H].

rotate_game(X) ->
    receive
        stop     -> io:format("~w game is over~n",[X]);
        {25, Ps} -> io:format("~w ends game~n",[X]),
                    lists:foreach( fun(P) -> P ! stop end, Ps);
        {N, Ps } -> io:format("~w increases ~w~n",[X, N]),
                    hd(Ps) ! {N + 1, rotate(Ps)},
                    rotate_game(X)
    end.
```

```
1> Ps = [spawn(lecture, rotate_game, [X]) || X <- lists:seq(1,10) ].
2> hd(Ps) ! {0, Ps}, ok.
```

Inhoud

Tips 'n tricks

- Met `halt().` of Ctrl+\sluit je de Erlang interpreter af.
- Met `f/1` kun je een variabele unbounden
- Met `f/0` kun je alle variabelen unbounden

Volgende week

Go!