JAKUB CZARNIECKI

FUNCTIONAL PROGRAMMING IN ERLANG

A SHORT NOTE ABOUT

FUNCTIONAL PROGRAMMING

THE BASICS

DATA TYPES IN ERLANG

NUMBERS

```
1> 2 + 15.
17
2> 49 * 100.
4900
3> 1892 - 1472.
420
4> 5 / 2.
2.5
5> 5 div 2.
2
6> 5 rem 2.
1
```

ATOMS

```
1> atom.
atom
2> atoms_rule.
atoms_rule
3> atoms_rule@erlang.
atoms_rule@erlang
4> 'Atoms can be cheated!'.
'Atoms can be cheated!'
5> atom = 'atom'.
atom
```

TUPLES

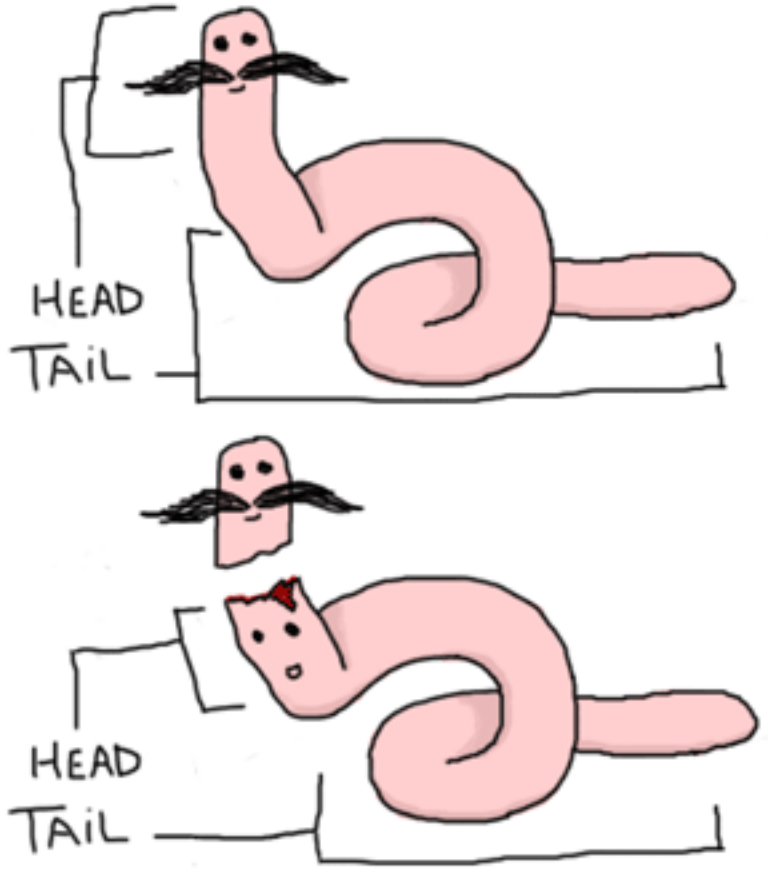
```
1> X = 10, Y = 4.
4
2> Point = {X,Y}.
{10,4}
```

```
3> Point = {4,5}.
{4,5}
4> {X,Y} = Point.
{4,5}
5> X.
4
6> {X,_} = Point.
{4,5}
```

```
7> {_,_} = {4,5}.
{4,5}
8> {_,_} = {4,5,6}.
** exception error: no match of right hand side value {4,5,6}
```

LISTS

```
1> [1, 2, 3, {numbers,[4,5,6]}, 5.34, atom].
[1,2,3,{numbers,[4,5,6]},5.34,atom]
2> [97, 98, 99].
"abc"
3> [97,98,99,4,5,6].
[97,98,99,4,5,6]
4> [233].
"é"
5> [1,2,3] ++ [4,5].
[1,2,3,4,5]
6> [1,2,3,4,5] -- [1,2,3].
[4,5]
7> [2,4,2] -- [2,4].
[2]
8> [2,4,2] -- [2,4,2].
[]
```



BINARIES

```
1> Color = 16#F09A29.
15768105
2> Pixel = <<Color:24>>.
<<240,154,41>>
```

```
3> Pixels = <<213,45,132,64,76,32,76,0,0,234,32,15>>.
<<213,45,132,64,76,32,76,0,0,234,32,15>>
4> <<Pix1,Pix2,Pix3,Pix4>> = Pixels.
** exception error: no match of right hand side value
<<213,45,132,64,76,32,76,
                                                              0,0,234,32,15>>
5> <<Pix1:24, Pix2:24, Pix3:24, Pix4:24>> = Pixels.
<<213,45,132,64,76,32,76,0,0,234,32,15>>
```

```
6> <<R:8, G:8, B:8>> = <<Pix1:24>>.
<<213,45,132>>
7> R.
213
```

```
8> <<R:8, Rest/binary>> = Pixels.
<<213,45,132,64,76,32,76,0,0,234,32,15>>
9> R.
213
```

MAPS

```
1> Pets = #{"dog" => "winston", "fish" => "mrs.blub"}.
#{"dog" => "winston","fish" => "mrs.blub"}
2> #{"fish" := CatName, "dog" := DogName} = Pets.
#{"dog" => "winston","fish" => "mrs.blub"}
```

```
3> Pets#{"dog" := "chester"}.
#{"dog" => "chester","fish" => "mrs.blub"}
4> Pets#{dog := "chester"}.
** exception error: bad argument
     in function  maps:update/3
        called as maps:update(dog,"chester",#{"dog" => "winston","fish" =>
        "mrs.blub"})
     in call from erl_eval:'-expr/5-fun-0-'/2 (erl_eval.erl, line 257)
     in call from lists:foldl/3 (lists.erl, line 1248)
```

```
5> #{"favorite" := Animal, Animal := Name} = Pets#{"favorite" := "dog"}.
#{"dog" => "winston","favorite" => "dog","fish" => "mrs.blub"}
6> Name.
"winston"
```

PATTERN MATCHING!

BASIC EXAMPLES

```
1> Humidity = {percent, 90}.
{percent,90}
2> {percent, P} = Humidity.
{percent,90}
3> P.
90
```

```
1> Elements = [first, second, third].
[first,second,third]
2> [F | [S | [T | _]]] = Elements.
[first,second,third]
3> {F, S, T}.
{first,second,third}
```

PATTERN MATCHING IN FUNCTIONS

```
1> HTTPError = fun
1>       (bad_request) -> 400;
1>       (not_found) -> 404;
1>       (internal_server_error) -> 500
1> end.
#Fun<erl_eval.6.90072148>
2>
2> HTTPError(not_found).
404
3> HTTPError(forbidden).
** exception error: no function clause matching
                    erl_eval:'-inside-an-interpreted-fun-'(forbidden)
```

MATCH IPV4 HEADER IN ONE LINE

```
1> Packet = <<16#45, 16#00, 16#00, 16#44, 16#ad, 16#0b, 16#00, 16#00, 16#40,
16#11, 16#72, 16#72, 16#ac, 16#14, 16#02, 16#fd, 16#ac, 16#14, 16#00, 16#06>>.
<<69,0,0,68,173,11,0,0,64,17,114,114,172,20,2,253,172,20, 0,6>>

2> <<Version:4, IHL:4, TypeOfService:8, TotalLength:16,  Identification:16,
FlagX:1, FlagD:1, FlagM:1,  FragmentOffset:13, TTL:8, Protocol:8,
HeaderCheckSum:16, SourceAddress:32, DestinationAddress:32, Rest/binary>> = Packet.
<<69,0,0,68,173,11,0,0,64,17,114,114,172,20,2,253,172,20, 0,6>>

3> Version.
4
```

THE BASICS

FUNCTIONS

SIMPLE LIST SUMATOR

```
sumator(List) ->
    sumator(0, List).

sumator(Sum, []) -> Sum;
sumator(Sum, [Next | Rest]) ->
    sumator(Sum + Next, Rest).

4> s:sumator([]).
0
5> s:sumator([1,2,3,4,5,6]).
21
6> s:sumator(0).
** exception error: no function clause matching s:sumator(0,0) (s.erl, line 7)
```

WHAT ARE

PURE FUNCTIONS

```erlang
fibonacci(0) -> 0;
fibonacci(1) -> 1;
fibonacci(N) -> fibonacci(N - 1) + fibonacci(N - 2).


fibonacci2(0) -> 0;
fibonacci2(1) -> 1;
fibonacci2(N) -> fibonacci2(N - 2, 0, 1).

fibonacci2(0, N2, N1) -> N1 + N2;
fibonacci2(Left, N2, N1) -> fibonacci2(Left - 1, N1, N1 + N2).
```

THE IMPORTANCE OF THE

TAIL CALL (TCO)

WHAT IS A TAIL CALL AND WHY IT'S IMPORTANT?

- Tail call is when the last expression in a function is the function call.
- Function which calls itself in the last expression is said to be *tail-recursive*.
- Erlang does optimise tail-recursive functions by replacing function arguments and jumping to the beginning of the function drastically reducing recursion overhead.
- No stack overflows.

Naive version took almost 1 minute for fib(45)!

```
fibonacci(0) -> 0;
fibonacci(1) -> 1;
fibonacci(N) -> fibonacci(N - 1) + fibonacci(N - 2).

2> {Time, Result} = timer:tc(fib, fibonacci, [45]).
{54274744,1134903170}
3> Time div 1000000.
54
```

Optimised version took 1 microsecond for fib(45)

```
fibonacci2(0) -> 0;
fibonacci2(1) -> 1;
fibonacci2(N) -> fibonacci2(N - 2, 0, 1).

fibonacci2(0, N2, N1) -> N1 + N2;
fibonacci2(Left, N2, N1) -> fibonacci2(Left - 1, N1, N1 + N2).

2> {Time, Result} = timer:tc(fib, fibonacci2, [45]).
{1,1134903170}
3> Time.
1
```

… And only 10 seconds for fib(1000000)!

```
fibonacci2(0) -> 0;
fibonacci2(1) -> 1;
fibonacci2(N) -> fibonacci2(N - 2, 0, 1).

fibonacci2(0, N2, N1) -> N1 + N2;
fibonacci2(Left, N2, N1) -> fibonacci2(Left - 1, N1, N1 + N2).

2> {Time, Result} = timer:tc(fib, fibonacci2, [1000000]).
{10252150,_}
3> Time div 1000000.
10
```

WHAT DOES IT MEAN TO HAVE

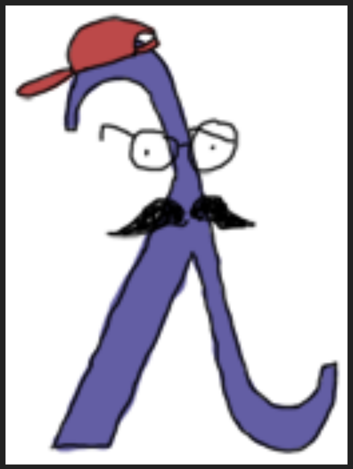
FIRST-CLASS FUNCTIONS

Passing functions as arguments to other functions

```
7> First20 = lists:map(fun fib:fibonacci2/1, lists:seq(1, 20)).
[1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,
 4181,6765]
8> IsEven = fun(N) -> N rem 2 == 0 end.
#Fun<erl_eval.6.90072148>
9> EvenOnly = lists:filter(IsEven, First20).
[2,8,34,144,610,2584]
```

Returning functions as the values from other functions

```
19> GenerateFibs = fun(N) ->
19>     fun() ->
19>         lists:map(fun fib:fibonacci2/1, lists:seq(1, N))
19>     end
19> end.
#Fun<erl_eval.6.90072148>
20> GenerateFirst100 = GenerateFibs(100).
#Fun<erl_eval.20.90072148>
21> GenerateFirst100().
[1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,
 4181,6765,10946,17711,28657,46368,75025,121393,196418,
 317811,514229|...]
```

AND WHAT ARE 

HIGHER-ORDER FUNCTIONS

AN IMPORTANT PART OF ALL FUNCTIONAL PROGRAMMING LANGUAGES IS THE ABILITY TO TAKE A FUNCTION YOU DEFINED AND THEN PASS IT AS A PARAMETER TO ANOTHER FUNCTION. THIS IN TURN BINDS THAT FUNCTION PARAMETER TO A VARIABLE WHICH CAN BE USED LIKE ANY OTHER VARIABLE WITHIN THE FUNCTION. A FUNCTION THAT CAN ACCEPT OTHER FUNCTIONS TRANSPORTED AROUND THAT WAY IS NAMED A HIGHER ORDER FUNCTION. HIGHER ORDER FUNCTIONS ARE A POWERFUL MEANS OF ABSTRACTION AND ONE OF THE BEST TOOLS TO MASTER IN ERLANG.

```
29> lists:map(fun(X) -> {X, fib:fibonacci2(X)} end, lists:seq(0, 20)).
[{0,0},
 {1,1},
 {2,1},
 {3,2},
 {4,3},
 {5,5},
 {6,8},
 {7,13},
 {8,21},
 {9,34},
 {10,55},
 {11,89},
 {12,144},
 {13,233},
 {14,377},
 {15,610},
 {16,987},
 {17,1597},
 {18,2584},
 {19,4181},
 {20,6765}]
```


Side
Effects

WHY IT IS GOOD TO LIMIT SIDE EFFECTS?

- Code is easier to test – the pure function always returns the same output for the same input.
- Code is easier to read and maintain - there are no hidden dependencies.

IMMUTABLE DATA.

HOW DATA IS REPRESENTED IN ERLANG?

- Every time you pass the data over to another function, it is copied over (there is no passing by reference)
- Erlang VM can make assumptions based on guarantee of immutability of the data and optimise access (copy-on-write, only write new data)

IDEMPOTENCE.

WHAT IS IDEMPOTENCE?

- Function applied twice for the same value gives the same result as if it was applied only once.
  $f(f(x)) \equiv f(x)$
- This term also applies to wider topic like composition of functions - every single function in the chain can be idempotent but the composition as a whole may not be idempotent.

NOT MODIFYING THE DATA AT ALL

NULLIPOTENT FUNCTIONS

HELLO, OUTSIDE WORLD

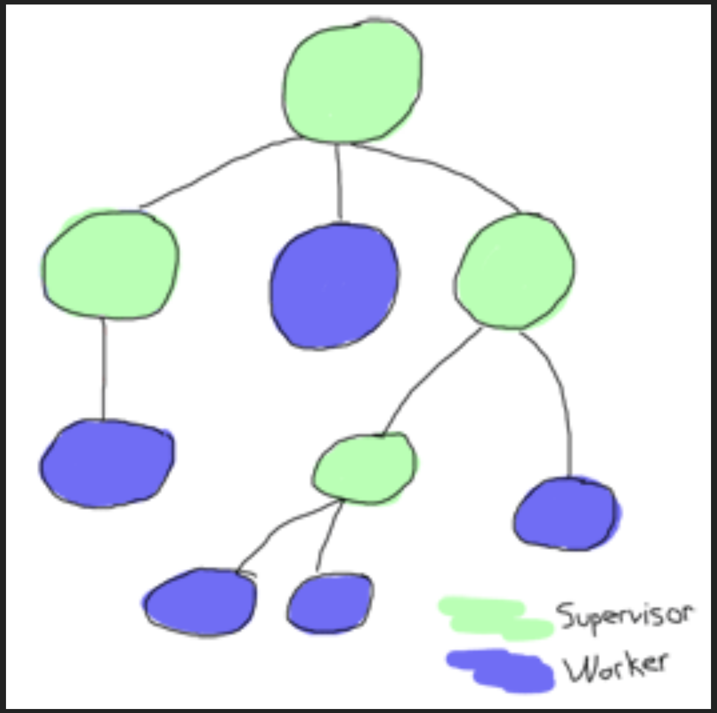
HOW DO YOU KEEP SOME STATE THEN?

- In Erlang the way to keep the state is to use processes.
- If we need to share the state between processes and do it often - it might be beneficial to use ETS.
- Mnesia for keeping more data in a cluster while still using built-in Erlang functionality.
- External database / cache storage.

LET IT CRASH

FAULT TOLERANCE IN ERLANG

- Crash of a process does not mean a crash for the whole VM.
- Erlang/OTP supervision trees - process that got crashed will be restarted by their supervisor.
- Thanks to these mechanisms you do not need to program defensively and can deliver more robust solution without much of error handling code.



A SHORT TALK ABOUT

SUPERVISORS

TRACING AND DEBUGGING LIVE

```
10> dbg:tracer().
{ok,<0.50.0>}
11> dbg:p(all,c).
{ok,[{matched,nonode@nohost,26}]}
12> dbg:tpl(s, x).
{ok,[{matched,nonode@nohost,4},{saved,x}]}
13> s:sumator([1,2,3,4,10]).
(<0.32.0>) call s:sumator([1,2,3,4,10])
(<0.32.0>) call s:sumator(0,[1,2,3,4,10])
(<0.32.0>) call s:sumator(1,[2,3,4,10])
(<0.32.0>) call s:sumator(3,[3,4,10])
(<0.32.0>) call s:sumator(6,[4,10])
(<0.32.0>) call s:sumator(10,"\n")
(<0.32.0>) call s:sumator(20,[])
(<0.32.0>) returned from s:sumator/2 -> 20
(<0.32.0>) returned from s:sumator/2 -> 20
(<0.32.0>) returned from s:sumator/2 -> 20
(<0.32.0>) returned from s:sumator/2 -> 20
(<0.32.0>) returned from s:sumator/2 -> 20
(<0.32.0>) returned from s:sumator/2 -> 20
(<0.32.0>) returned from s:sumator/1 -> 20
20
```

QUESTIONS?

THANK YOU

CONTACT ME ON

JAKUBC@QBURST.COM