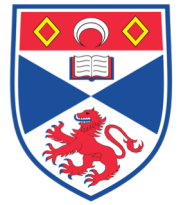

Introduction to (sequential) Erlang

Chris Brown
(thanks to Tamas Kozsik, ELTE)
University of St Andrews

cmb21@st-andrews.ac.uk

A brief history of Erlang

- Created in 1986 and fully open-sourced in 1996
- Designed by Ericsson (“Ericsson Language”)
  - Joe Armstrong
  - Robert Virding
  - Mike Williams
- Telephony switching

What is Erlang?

- Functional language
- Strict
- Dynamically typed
- Garbage collected
- Designed for concurrency and distribution
  - “Lightweight” processes
    - (which are not really “lightweight”)
  - Actor Model
- Processing binary data
- Fault tolerance (i.e. failure recovery)
- Compiles to beam, runs in a VM
- Hot code swap

Terms

- Literals
  - `42` or `42.0`
- Atoms
  - `leaf, blue, ok, error, true`
- Functions
  - `fun(X) -> X+1 end`
- Lists
  - `[0,1,1,2,3,5,8,13,21]`
- Tuples
  - `{may, 10, 2014}`
- Records
  - `#date{month=may, day=10, year=2014}`
- Binaries
  - `<<0,1,1,2,3,5,8,13,21>>`
- Pids, ports, refs

Expressions

- Terms (literals, atoms, compound data)
- Variables
  - `X, Long_Variable_Name`
- Function/operator calls
  - `fib(N-1)+fib(N-2)`
- Data structures
  - `{may, Day, fib(18)-570}`
- Control structures
  - Branching (case and if)
  - Sending and receiving a message
  - Error handling

Functions

```
increment(N) -> N+1.
```

Case Statements

```
fib(N) ->
     case N of
          0 -> 0;
          1 -> 1;
          _ -> fib(N-1) + fib(N+2).
```

Function clauses

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

Guards

```
fib(N) when N < 2 -> N;
fib(N) -> fib(N-1) + fib(N-2).
```

If expressions

```
fib(N) ->
    if
        N < 2 -> N;
        true  -> fib(N-1) + fib(N-2)
    end.
```

Recursion

```
factorial(1) -> 1;
factorial(N) -> N * factorial(N-1).
```

- factorial(3)
  matches N = 3 in clause 2
  == 3 * factorial(3 - 1)
  == 3 * factorial(2)
  matches N = 2 in clause 2
  == 3 * (2 * factorial(2 - 1))
  == 3 * (2 * factorial(1))
  matches clause 1
  == 3 * (2 * 1)
  == 3 * 2
  == 6

(from An Erlang Course, http://www.erlang.org/course/course.html)

Tail recursion

```
factorial(1) -> 1;
factorial(N) -> N * factorial(N-1).

-----------------------------------

factorial(N) -> factorial_acc(N,1).

factorial_acc(1,Acc) -> Acc;
factorial_acc(N,Acc) ->
            factorial_acc(N-1, Acc*N).
```

More tail recursion

```
prime(1) -> false;
prime(N) when N > 1 -> prime(N,2).

% no proper divisors of N between M and sqrt(N)
prime(N,M) when M*M>N -> true;
prime(N,M) -> (N rem M =/= 0) andalso
                                prime(N,M+1).
```

Pattern matching

```
fib(N) ->
    case N of 0 -> 0;
      1 -> 1;
      _ -> fib(N-1) + fib(N-2)
    end.
```

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

Lists

```
[0,1,1,2,3,5,8,13,21]

[0 | [1,1,2,3,5,8,13,21]]

[0 | [1 | [1 | [2 | [3 | [5 | [8 | [13 | [21 |[]]]]]]]]]]

[0,1,1,2,3 | [5 | [8 | [13 | [21 | []]]]]]

[0,1,1,2,3 | [5,8,13,21 ]]
```

Recursion over lists

- Linear data structure, just like in Haskell
- Head and tail [Head | Tail]
- Recursive structure

```
sum([]) -> 0;
sum([Head|Tail]) -> Head + sum(Tail).
```

List comprehensions

```
primes(S) -> [N || N<-S, prime(N)].
```

Higher-order functions

```
primes(S) -> [N || N<-S, prime(N)].

primes(S) -> filter(fun prime/1, S).

filter(Pred,List) ->
    [ Item || Item <- List, Pred(Item) ].
```

“Lambda” functions

```
mul(Scalar,List) ->
      map( fun(Item)->
               Scalar*Item end,
           List ).

map( fun(Item)->Scalar*Item end, List )
```

Variable binding

- Formal parameters:
  Example: `fib(N)-> ...`
  Example: `sum([Head|Tail])-> ...`

- Generator in list comprehension:

```
[ ... | Item <- List]
```

- Syntax: `Pattern = Expression`
  Example: `Primes = primes(List)`
  Example: `[Head|Tail] = List`

Sequencing expressions

```
area({square, Side}) -> Side * Side;
area({circle, Radius}) ->
      % almost :-)
      3 * Radius * Radius;
area({triangle, A, B, C}) ->
      S = (A + B + C)/2,
      math:sqrt(S*(S-A)*(S-B)*(S-C)).
```

Modules

- Place code into a .erl file
- Compile unit is a module

```
-module(mymath).
-export([fib/1,prime/1,pi/0]).
-define(PI,3.14).

pi() -> ?PI.

fib(N) when N<2 -> N;
fib(N) -> fib(N-1) + fib(N-2).

prime(1) -> false;
prime(N) when N > 1 -> prime(N,2).

prime(N,M) when M*M>N -> true;
prime(N,M) -> (N rem M =/= 0) andalso prime(N,M+1).
```

The Erlang REPL

```
$ ls mymath.erl mymath.erl
$ erl
Erlang R16B (erts-5.10.1) [source] [smp:4:4]
[async-threads:10] [hipe] [kernel-poll:false]
Eshell V5.10.1 (abort with ^G)
1> c(mymath).
{ok,mymath}
2> mymath:prime(1987).
true
3> q().
ok
4> $ ls mymath* mymath.beam mymath.erl
```

Thank you!

cmb21@st-andrews.ac.uk

@chrismarkbrown