Elementy języka **Erlang**

- Cechy języka
- Podstawy
- Programowanie sekwencyjne
- Programowanie współbieżne

Cechy języka

- równoległość na wielu rdzeniach
- odporność na awarie
- programowanie funkcyjne
- obliczenia czasu rzeczywistego
- obliczenia rozproszone
- hot swap

Podstawy

```
$ erl
Erlang/OTP…

Eshell… (abort with ^G)
1> 20+30.
50
2> 2+3*4.
14
3> (2+3)*4.
20
4> X = 123456789.
123456789
5> X.
123456789
6> X = 1234.
** exception error: no match of right side value 1234
```

zmienne nie są zmienne a = nie jest przypisaniem

Atomy

Ciąg znaków alfanumerycznych oraz _ i @ zaczynający się od małej litery.

```
> hello.
hello
```

Liczby całkowite

Dowolna precyzja.

```
> 16#cafe + 32#sugar.
30411865
```

Zmienne

Ciąg znaków alfanumerycznych zaczynający się od wielkiej litery.

Liczby zmiennopozycyjne

```
> 4/2.
2.0
> 5 div 3.
1
> Pi = 3.14159.
3.14159
> R = 5.
5
> Pi * R * R.
78.53975
```

Krotki

```
> Person = {person,
              {name, joe},
              {height, 1.82},
              {footsize, 42},
              {eyecolour, brown}}
{person, {name, joe}, {height, 1.82}, {footsize, 42},
 {eyecolour, brown}}
> F = {firstName, joe}
{firstName, joe}
> L = {lastName, armstrong}
{lastName, armstrong}
> P = {person, F, L}
{person, {firstName, joe}, {lastName, armstrong}}
> {true, Q, 23}.
* 1: variable 'Q' is unbound
```

Listy

```
> [1+7, hello, 2-2, {cost, apple, 30-20}, 3].
[8, hello, 0, {cost, apple, 10}, 3]

[]
[H | T]
```

Łańcuchy znaków

```
> Name = ”Hello”.
”Hello”
> [83, 117, 114, 112, 114, 105, 115, 101].
"Surprise"
```

Programowanie sekwencyjne

Moduły

```
%% geometry.erl
-module(geometry).
-export([area/1]).
area({rectangle, Width, Ht}) -> Width*Ht;
area({circle, R}) -> 3.14159*R*R.

> c(geometry)
{ok, geometry}
> geometry:area({rectangle, 10, 5}).
50
> geometry:area({circle, 1.4}).
6.15752
```

```
Pattern1 ->
    Expressions1;
Pattern2 ->
    Expressions2;
…
PatternN ->
    ExpressionsN.


%% lib_misc.erl
sum(L) -> sum(L, 0).
sum([], N) -> N;
sum([H | T], N) -> sum(T, H + N).
```

Lambda wyrażenia

```
> Z = fun(X) -> 2*X end.
#Fun<…>
> Z(2).
4
> Hypot = fun(X, Y) -> math:sqrt(X*X+Y*Y) end.
#Fun<…>
> Hypot(3, 4).
5.0
> Hypot(3).
** exception error: interpreted function with arity 2
called with one argument
> TempConvert = fun({c, C}) -> {f, 32+C*9/5};
                   ({f,F}) -> {c, (F-32)*5/9}
                end.
#Fun<…>
> L = [1,2,3,4].
[1,2,3,4]
> lists:map(Z, [1,2,3,4]).
[2,4,6,8]
```

```
> Even = fun(X) -> (X rem 2) =:= 0 end.
#Fun<…>
> lists:map(Even, [1,2,3,4,5,6,7,8]).
[false, true, false, true, false, true, false, true]
> lists:filter(Even, [1,2,3,4,5,6,7,8]).
[2,4,6,8]

> Mult = fun(Times) -> (fun(X) -> X*Times end) end.
#Fun<…>
> Triple = Mult(3).
#Fun<…>
> Triple(5).
15

%% lib_misc.erl
for(Max, Max, F) -> [F(Max)];
for(I, Max, F) -> [F(I) | for(I+1, Max, F)].

> lib_misc:for(1, 10, fun(I) -> I end).
[1,2,3,4,5,6,7,8,9,10]
> lib_misc:for(1, 10, fun(I) -> I*I end).
[1,4,9,16,25,36,49,64,81,100]


%% mylists.erl
sum([H | T]) -> H + sum(T);
sum([]) -> 0.

map(_, []) -> [];
map(F, [H | T]) -> [F(H) | map(F, T)].
```

Wyrażenia listowe

```
> [I*I || I <- [1,2,3]].
[1,4,9]    \__ generator __/

map(F, L) -> [F(X) || X <- L].

> [I || I <- [1,2,3,4,5], I rem 2 =:= 0].
[2,4]     \___ generator ___/      \_____ filtr _____/
```

Quicksort

```
qsort([]) -> [];
qsort([Pivot | T]) ->
    qsort([X || X <- T, X < Pivot])
    ++ [Pivot] ++
    qsort([X || X <- T, X >= Pivot]).
```

Trójki pitagorejskie

```
pythag(N) ->
    [{A, B, C} ||
        A <- lists:seq(1, N),
        B <- lists:seq(1, N),
        C <- lists:seq(1, N),
        A+B+C =< N,
        A*A + B*B =:= C*C
    ].
```

wyrażenie arytmetyczne	priorytet
+X	1
-X	1
X * Y	2
X / Y	2
bnot X	2
X div Y	2
X rem Y	2
X band Y	2
X + Y	3
X - Y	3
X bor Y	3
X bxor Y	3
X bsl N	3
X bsr N	3

Wartownicy

```
max(X, Y) when X > Y -> X;
max(X, Y) - Y.
```

`G1; G2; …; Gn` przynajmniej jeden spełniony
`G1, G2, …, Gn` wszystkie spełnione

```
f(X, Y) when is_integer(X), X > Y, Y < 6 -> …
```

```
f(X) when (X == 0) or (1/X > 2) ->
    …
```
wartownik zawodzi gdy X jest 0

```
g(X) when (X == 0) orelse (1/X > 2) ->
    …
```
wartownik spełniony gdy X jest 0

predykaty	funkcje
`is_atom(X)`	`abs(X)`
`is_binary(X)`	`element(N, X)`
`is_constant(X)`	`float(X)`
`is_float(X)`	`hd(X)`
`is_function(X)`	`length(X)`
`is_function(X, N)`	`node()`
`is_integer(X)`	`node(X)`
`is_list(X)`	`round(X)`
`is_number(X)`	`self()`
`is_pid(X)`	`size(X)`
`is_port(X)`	`trunc(X)`
`is_reference(X)`	`tl(X)`
`is_tuple(X)`	
`is_record(X, Tag)`	
`is_record(X, Tag, N)`	

Wyrażenia case i if

```
case Expression of
    Pattern1 [when Guard1] -> Expr_seq1;
    Pattern2 [when Guard2] -> Expr_seq2;
    …
end

filter(P, [H | T]) -> filter1(P(H), H, P, T);
filter(P, []) -> [].
filter1(true, H, P, T) -> [H | filter(P, T)];
filter1(false, H, P, T) -> filter(P, T).

filter(P, [H | T]) ->
    case P(H) of
        true -> [H | filter(P, T)];
        false -> filter(P, T)
    end;
filter(P, []) ->
    [].



if
    Guard1 -> Expr_seq1;
    Guard2 -> Expr_seq2;
    …
    true -> Expr_seqN
end
```

Aplikacja

```
apply(Mod, Func, [Arg1, Arg2, …, ArgN])
≡
Mod:Func(Arg1, Arg2, …, ArgN)
```

Operatory ++ i --

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

Programowanie współbieżne

- **Pid = spawn(Func)**
  tworzy nowy proces obliczający funkcję **Func**
- **Pid ! Message**
  wysyła komunikat do procesu o **Pid**
- **receive … end**
  odbiera i przetwarza komunikaty

```
receive
    Pattern1 [when Guard1] ->
        Expressions1;
    Pattern2 [when Guard2] ->
        Expressions2;
    …
end
```

```
%% area_server0.erl
-module(area_server0).
-export([loop/0]).

loop() ->
    receive
        {rectangle, Width, Ht} ->
          io:format("Area of rectangle is ~p~n",
                    [Width*Ht]),
          loop();
        {circle, R} ->
          io:format("Area of circle is ~p~n",
                    [3.14159*R*R]),
          loop();
        Other ->
          io:format("I dont know area of ~p~n",
                    [Other]),
          loop()
    end.
```

```
> c(area_server0).
> Pid = spawn(fun area_server0:loop/0).
> Pid ! {rectangle, 6, 10}.
Area of rectangle is 60
> Pid ! {circle, 1.2}.
Area of circle is 6.15752
> Pid ! {triangle, 4, 5, 6}.
I dont know area of {triangle, 4, 5, 6}
```

```
%% area_server1.erl
-module(area_server1).
-export([loop/0, rpc/2]).

loop() ->
    receive
        {From, {rectangle, Width, Ht}} ->
            From ! Wdith*Ht,
            loop();
        {From, {circle, R}} ->
            From ! 3.14159*R*R,
            loop();
        {From, Other} ->
            From ! {error, Other},
            loop()
    end.

rpc(Pid, Request) ->
    Pid ! {self(), Request},
    receive
        Response -> Response
    end.
```

Remote **P**rocedure **C**all

```
> c(area_server1).
> Pid = spawn(fun area_server1:loop/0).
> area_server1:rpc(Pid, {rectangle, 6, 10}).
60
> area_server1:rpc(Pid, {circle, 1.2}).
6.15752
> area_server1:rpc(Pid, {triangle, 4, 5, 6}).
{error, {triangle, 4, 5, 6}}
```

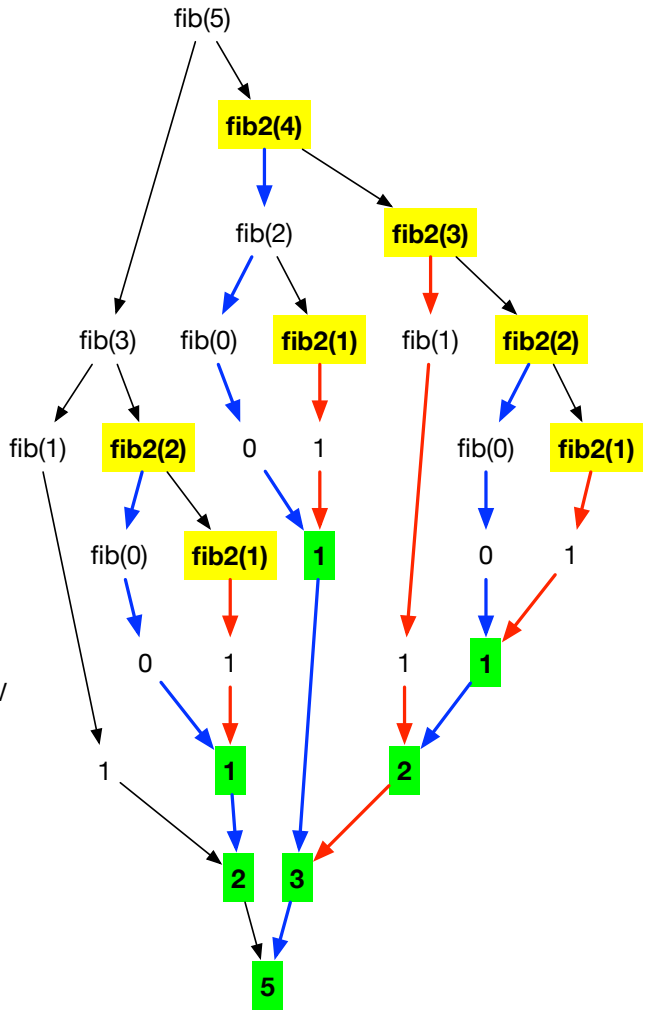
```
%% fib2.erl
-module(fib2).
-export([fib/1]).

fib(0) -> 0;
fib(1) -> 1;
fib(N) ->
    fib2(N - 1) + fib(N - 2).

fib2(N) ->
    Pid = spawn(fun worker/0),
    Pid ! {self(), N},
    receive
        F -> F
    end.

worker() ->
    receive
        {From, N} -> From ! fib(N)
    end.
```

```
> c(fib2).
{ok, fib}
> fib2:fib(5).
5
> fib2:fib(30).
832040
> fib2:fib(40).
102334155
```

7 procesów

1,346,268 procesów

165,580,140 procesów



Serwer par klucz-wartość

put(Key, Value) -> OldValue | undefined

zapisuje w słowniku parę klucz-wartość

get(Key) -> Value | undefined

pobiera ze słownika wartość związaną z kluczem

register(AnAtom, Pid)

rejestruje Pid procesu pod nazwą AnAtom

unregister(AnAtom)

usuwa rejestrację związaną z AnAtom

whereis(AnAtom) -> Pid | undefined

znajduje gdzie jest zarejestrowany AnAtom

rpc:call(Node, Mod, Fun, [Arg1, Arg2, …, ArgN])

```
%% kvs.erl
-module(kvs).
-export([start/0, store/2, lookup/1]).

start() -> register(kvs, spawn(fun() -> loop() end)).

store(Key, Value) -> rpc({store, Key, Value}).

lookup(Key) -> rpc({lookup, Key}).

rpc(Q) ->
   kvs ! {self(), Q},
   receive
      {kvs, Reply} -> Reply
   end.

loop() ->
   receive
      {From, {store, Key, Value}} ->
         put(Key, {ok, Value}),
         From ! {kvs, true},
         loop();
      {From, {lookup, Key}} ->
         From ! {kvs, get(Key)},
         loop()
   end.
```

co gdy nie można doczekać się na odpowiedź?

Przykład 1: prosty serwer nazw

```
$ erl
> kvs:start().
true
> kvs:store({location, joe}, ”Stockholm”).
true
> kvs:store(weather, raining).
true
> kvs:lookup(weather).
{ok, raining}
> kvs:lookup({location, joe}).
{ok, ”Stockholm”}
> kvs:lookup({location, jane}).
undefined
```

Przykład 2: klient i serwer na tym samym serwerze ale różnych węzłach

```
$ erl -sname gandalf
(gandalf@localhost)> kvs:start().
true
```

```
$ erl -sname bilbo
(bilbo@localhost)> rpc:call(gandalf@localhost, kvs,
                            store, [weather, fine]).
true
(bilbo@localhost)> rpc:call(gandalf@localhost, kvs,
                            lookup, [weather]).
{ok,fine}
```

Przykład 3: klient i serwer na różnych serwerach ale w tej samej podsieci

```
doris $ erl -name gandalf -setcookie abc
(gandalf@doris.domain)> kvs:start().
true
```

```
george $ erl -name bilbo -setcookie abc
(bilbo@george.domain)> rpc:call(gandalf@doris.domain,
                                kvs,
                                store, [weather, fine]).
true
(bilbo@george.domain)> rpc:call(gandalf@doris.domain,
                                kvs,
                                lookup, [weather]).
{ok,fine}
```

Timeout

```
receive
  …
after Time ->
  …
end
```

czas w milisekundach

```
sleep(T) ->
    receive
    after T -> ok
    end.

flush() ->
    receive
        _ -> flush()
    after 0 -> ok
    end.
```

Obliczenia na wielu rdzeniach

```
przemko@otryt:~$ erl
Erlang/OTP 18 [erts-7.3] [source] [64-bit] [smp:80:80]
[async-threads:10] [kernel-poll:false]

Eshell V7.3  (abort with ^G)
1>
```

timer:tc(Mod, Func, Args) -> {Time, Value}

czas w mikrosekundach

catch(Expr)

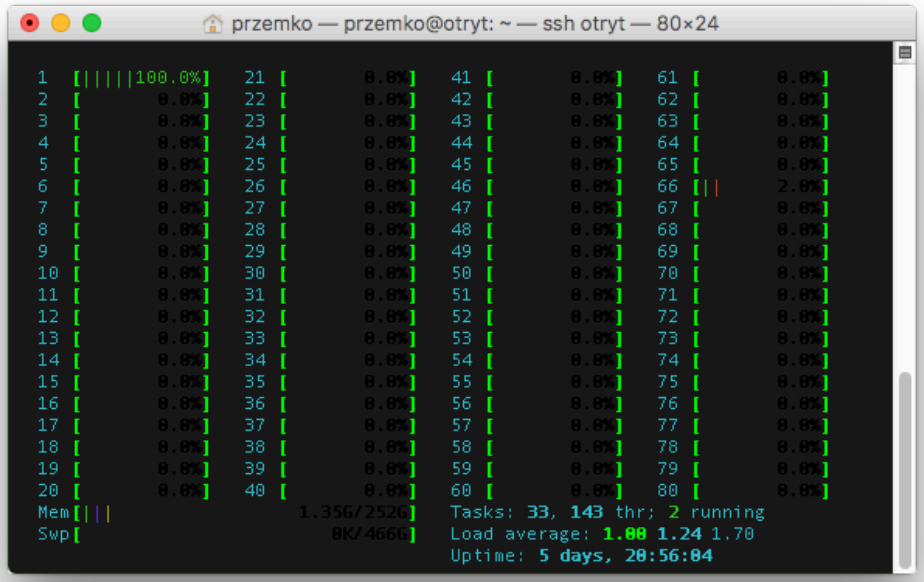
w przypadku błędu ma wartość **{'EXIT', opis_wyjątku}**

```
> A = 1/2.
0.5
5> B = 1/0.
** exception error: an error occurred when evaluating an arithmetic
expression
     in operator  '/'/2
        called as 1 / 0
6> catch(C = 1/2).
0.5
7> catch(D = 1/0).
{'EXIT',{badarith,[{erlang,'/',[1,0],[]},
                   {erl_eval,do_apply,6,[{file,"erl_eval.erl"},{line,674}]},
                   {erl_eval,expr,5,[{file,"erl_eval.erl"},{line,438}]},
                   {erl_eval,expr,5,[{file,"erl_eval.erl"},{line,431}]},
                   {shell,exprs,7,[{file,"shell.erl"},{line,686}]},
                   {shell,eval_exprs,7,[{file,"shell.erl"},{line,641}]},
                   {shell,eval_loop,3,[{file,"shell.erl"},{line,626}]}]}}
```

Przykład: równoległy map

```
-module(lib_misc).
-export([map/2, pmap/2]).

map(_, []) -> [];
map(F, [H|T]) -> [F(H)|map(F, T)].

pmap(F, L) ->
   S = self(),
   Ref = erlang:make_ref(), % świeża wartość
   Pids = map( fun(I) ->
            spawn(fun() -> do_f(S, Ref, F, I) end)
         end, L),
   gather(Pids, Ref).

do_f(Parent, Ref, F, I) ->
   Parent ! {self(), Ref, (catch F(I))}.

gather([Pid | T], Ref) ->
   receive
      {Pid, Ref, Ret} -> [Ret|gather(T, Ref)]
   end;
gather([], _) ->
   [].
```

```
> timer:tc(lib_misc, map, [fun fib:fib/1,
                           [40||_<-lists:seq(1,80)]]).
{1609146911,                                                    26,8 minut
 [102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155|...]}
```




```
> timer:tc(lib_misc, pmap, [fun fib:fib/1,
                            [40||_<-lists:seq(1,80)]]).
{30516393,                                                      30.5 sekundy
 [102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155,102334155,102334155,
  102334155,102334155,102334155,102334155|...]}
```