IMPLEMENTAREA CONCURENTEI IN LIMBAJE DE PROGRAMARE

Ioana Leustean

INTRODUCERE IN

ERLANG

http://www.erlang.org/

PARALELISM

CONCURENTA

SISTEME DISTRIBUITE

"Erlang was designed from the bottom up to program concurrent, distributed, fault-tolerant, scalable, soft, real-time systems. [...]
If your problem is concurrent, if you are building a multiuser system, or if you are building a system that evolves with time, then using Erlang might save you a lot of work, since Erlang was explicitly designed for building such systems. [...]
Processes interact by one method, and one method only, by exchanging messages. Processes share no data with other processes. This is the reason why we can easily distribute Erlang programs over multicores or networks. "
Joe Armstrong, Programming Erlang, Second Edition 2013

➢Bibliografie

Joe Armstrong, Robert Virding, Mike Williams, Concurrent Programming in Erlang, 1993

Joe Armstrong, Programming Erlang, Second Edition 2013

Fred Hébert, Learn You Some Erlang For Great Good, 2013

https://www.erlang.org/doc/

- Erlang este dezvoltat de Ericsson (initial in 1986)
  Creatorii: Joe Armstrong, Robert Virding, and Mike Williams

- Erlang este un limbaj functional
  Nu are variabile mutabile.
  Are functii de nivel inalt.
  Sistemul tipurilor este dinamic, verificarea corectitudinii se face la rulare.

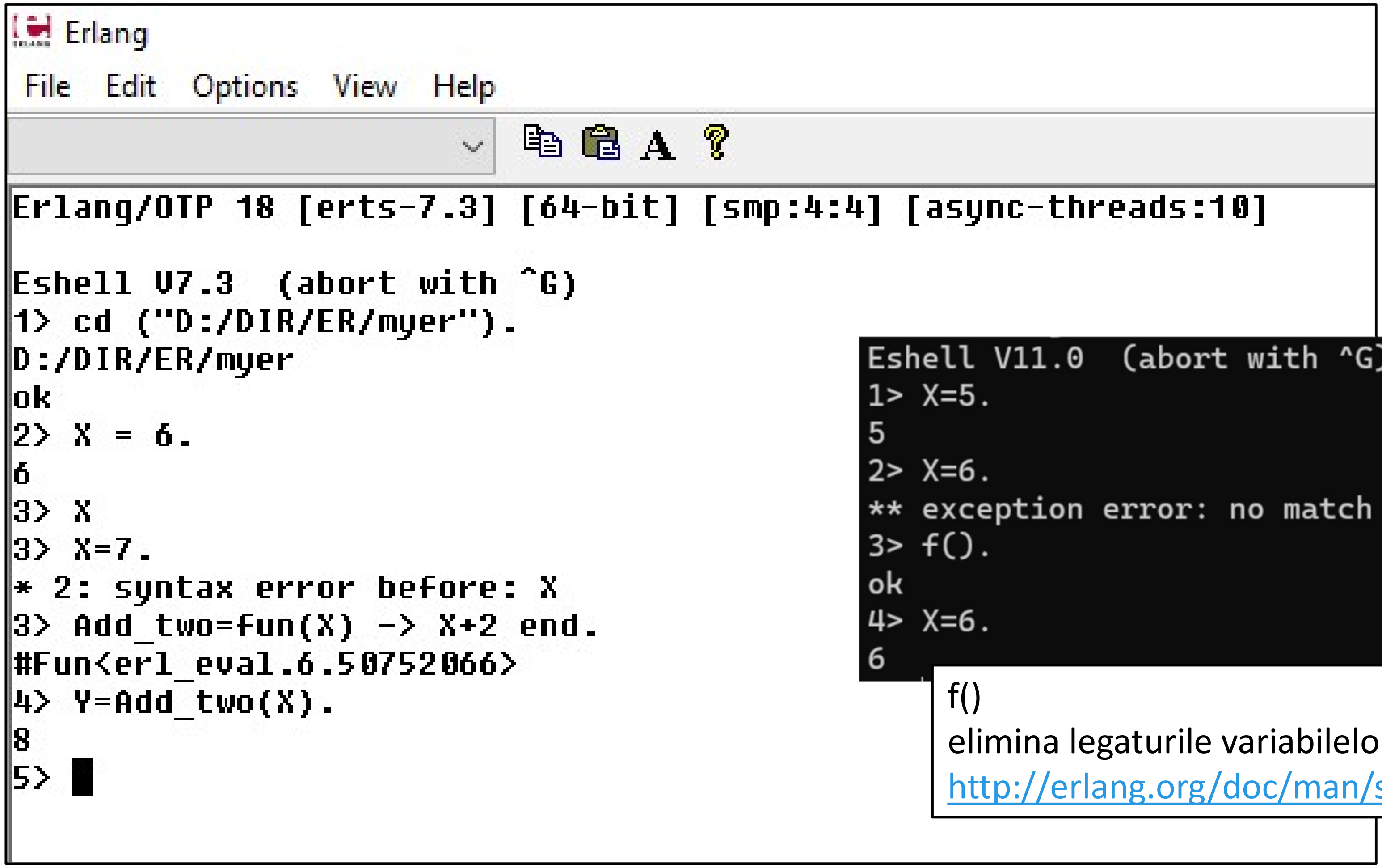
- Codul este compilat si rulat pe o masina virtuala numita BEAM.
- Erlang/OTP (Open Telecom Platform)
  OTP este o multime de librarii si tool-uri folosite pentru a crea aplicatii distribuite

Numele vine de la
- ❖ Agner Krarup Erlang (1878-1929) matematician si inginer danez
- ❖ **Er**icsson **Lang**uage

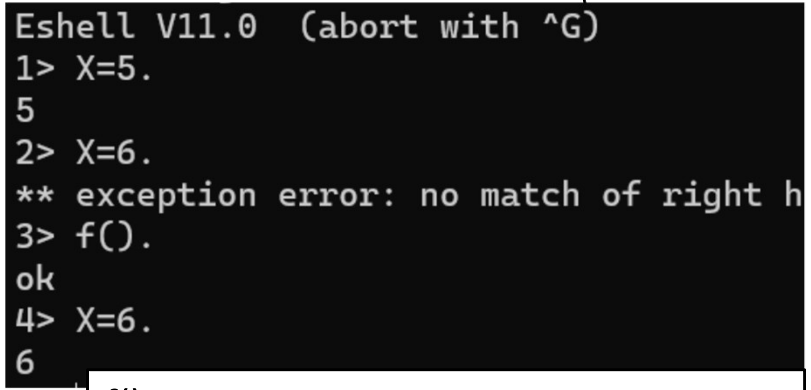
Limbajul Erlang - introducere

werl

http://erlang.org/doc/man/shell.html



```
Erlang/OTP 18 [erts-7.3] [64-bit] [smp:4:4] [async-threads:10]

Eshell V7.3  (abort with ^G)
1> cd ("D:/DIR/ER/myer").
D:/DIR/ER/myer
ok
2> X = 6.
6
3> X
3> X=7.
* 2: syntax error before: X
3> Add_two=fun(X) -> X+2 end.
#Fun<erl_eval.6.50752066>
4> Y=Add_two(X).
8
5> 
```



```
Eshell V11.0  (abort with ^G)
1> X=5.
5
2> X=6.
** exception error: no match of right h
3> f().
ok
4> X=6.
6
```

f()
elimina legaturile variabilelor
http://erlang.org/doc/man/shell.html

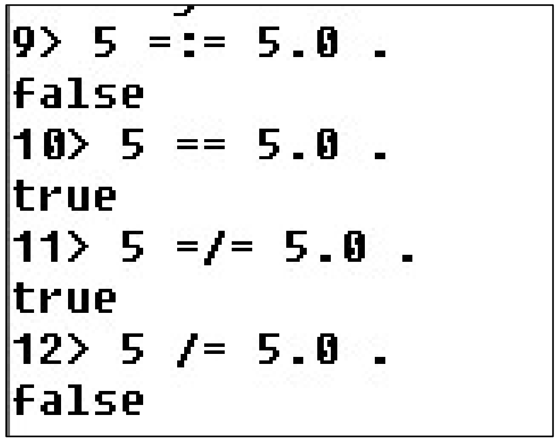
Erlang

- Comentariile incep cu %
  % comentariu pe o linie
- Variabilele incep cu litera mare [sau _] (celelalte caractere sunt alfanumerice, @,_)
- atomii incep cu litera mica; numele functiilor sunt atomi
- termen = data de orice tip
- orice instructiune se termina cu punct .
- un program este format din module;
  numele fisierului coincide cu numele modulului si are extensia .erl;
  compilarea se face folosind comanda c(nume_fisier)

Tipuri de date

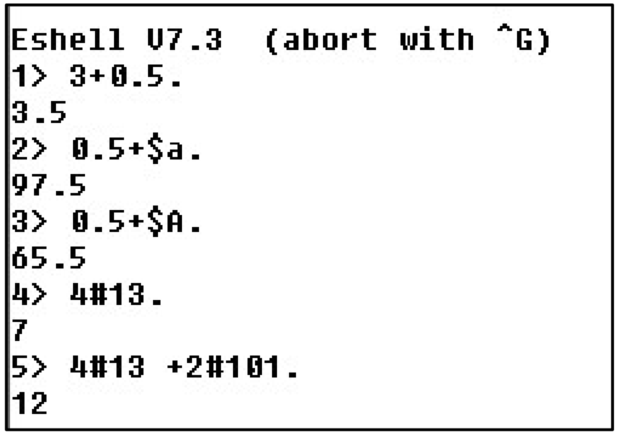
- Number:

Integer



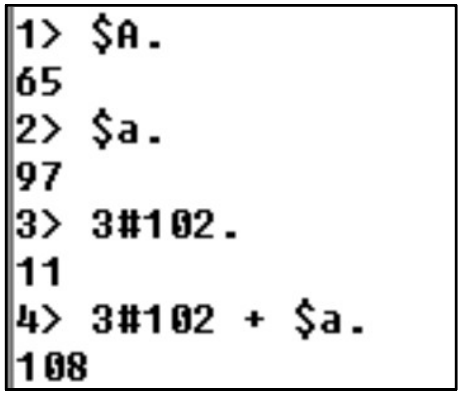
```
9> 5 =:= 5.0 .
false
10> 5 == 5.0 .
true
11> 5 =/= 5.0 .
true
12> 5 /= 5.0 .
false
```

Floats



```
Eshell V7.3  (abort with ^G)
1> 3+0.5.
3.5
2> 0.5+$a.
97.5
3> 0.5+$A.
65.5
4> 4#13.
7
5> 4#13 +2#101.
12
```

$char % codul ASCII
base#integer

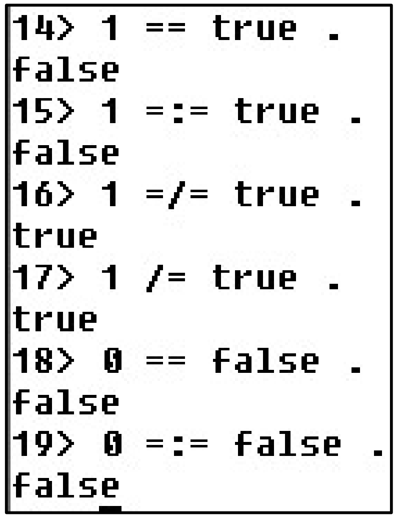


```
1> $A.
65
2> $a.
97
3> 3#102.
11
4> 3#102 + $a.
108
```

http://erlang.org/doc/reference_manual/data_types.html

Tipuri de date

- Boolean



```
14> 1 == true .
false
15> 1 =:= true .
false
16> 1 =/= true .
true
17> 1 /= true .
true
18> 0 == false .
false
19> 0 =:= false .
false
```

orelse/andalso

Expr1 **orelse** Expr2
Expr1 **andalso** Expr2

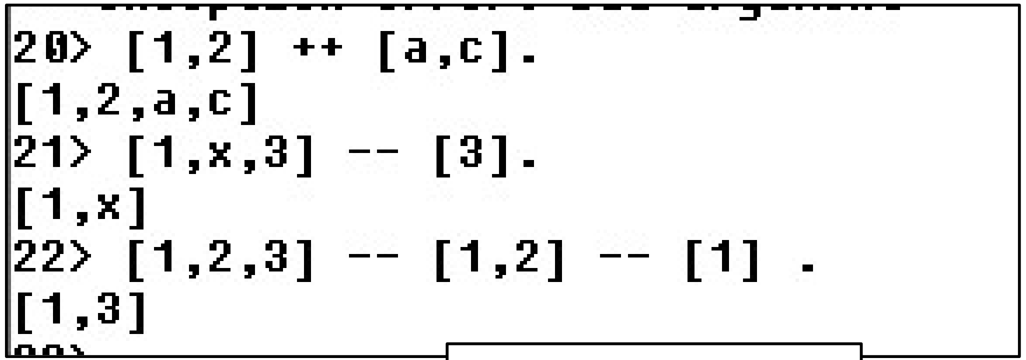
al doilea argument este evaluat numai la nevoie

- Atoms (named symbolic constants)
  luni, 'Luni', 'Prima zi'

http://erlang.org/doc/reference_manual/data_types.html

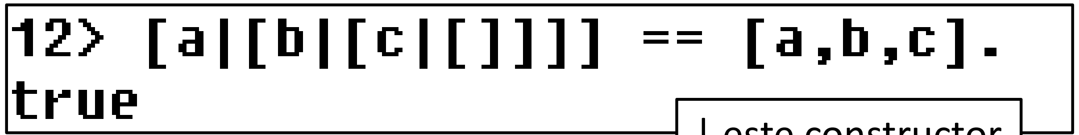
Tipuri de date

- Liste



```
20> [1,2] ++ [a,c].
[1,2,a,c]
21> [1,x,3] -- [3].
[1,x]
22> [1,2,3] -- [1,2] -- [1] .
[1,3]
```

--, ++ right-associative



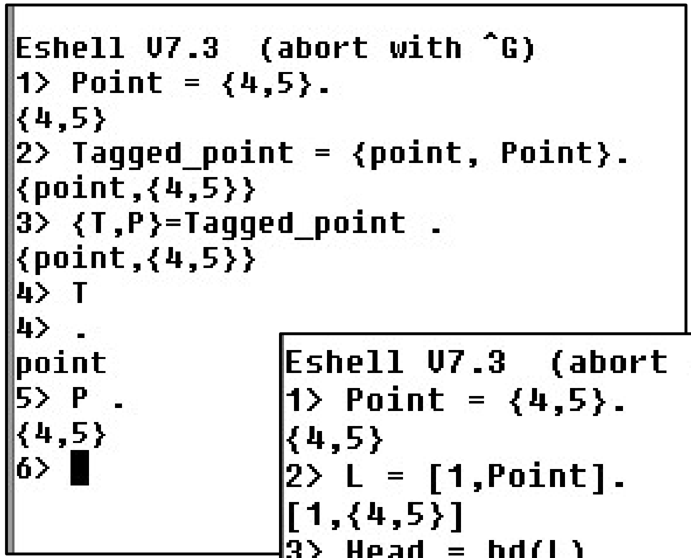
```
12> [a|[b|[c|[]]]] == [a,b,c].
true
```

| este constructor

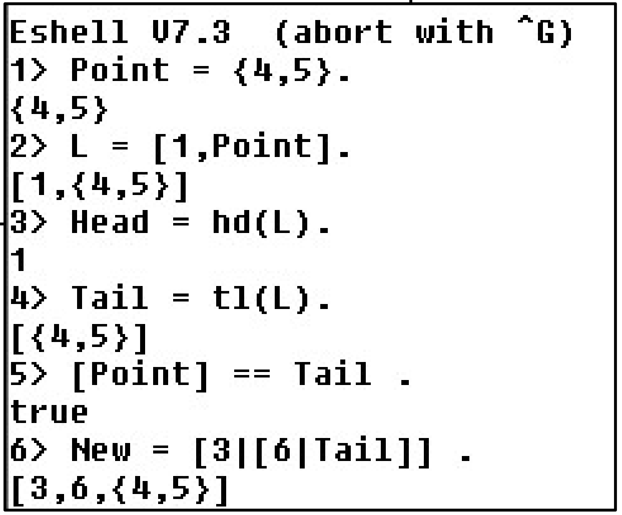
[1,2,a,c]

listele pot avea elemente de tipuri diferite

- Tupluri

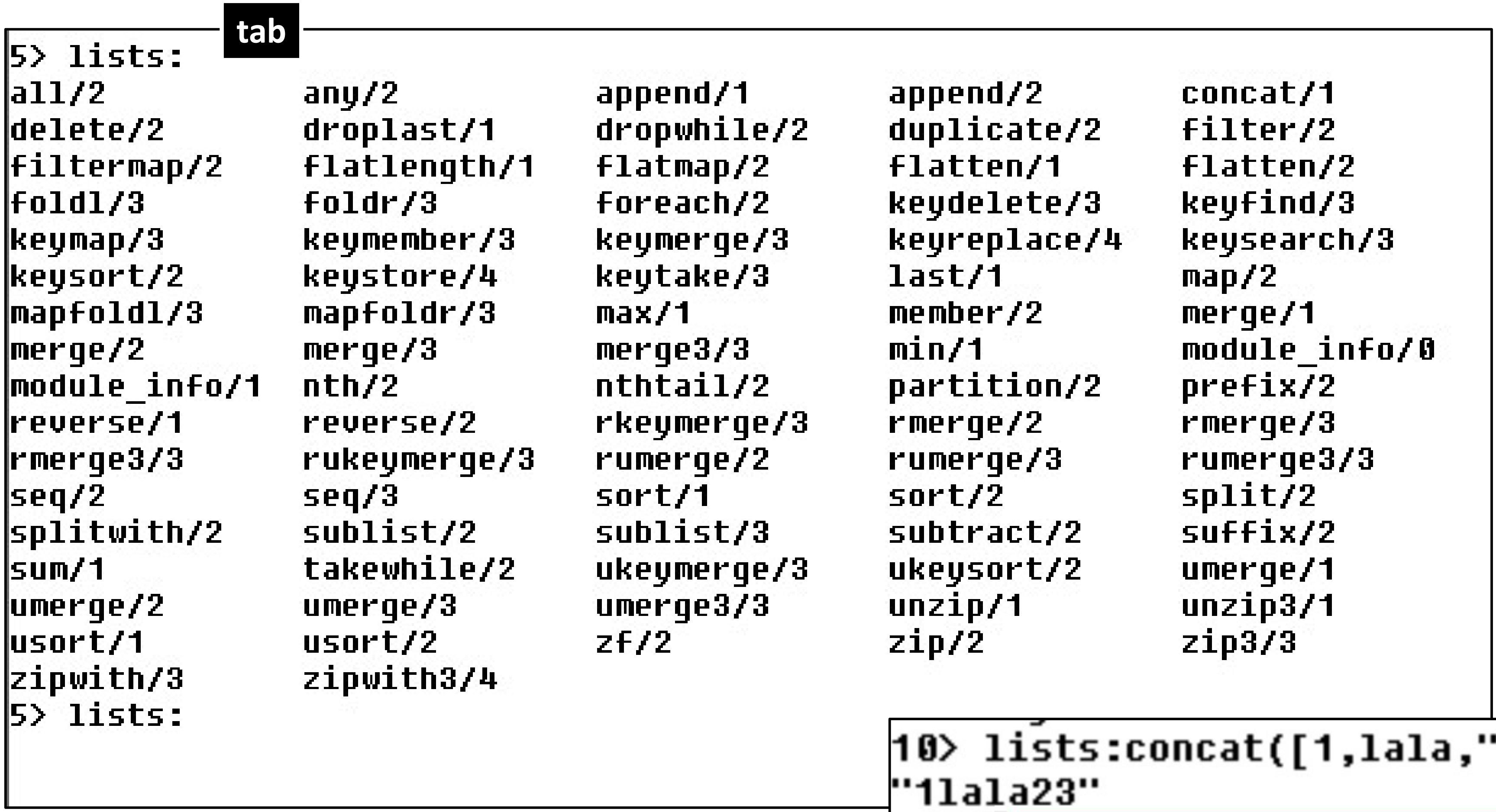


```
Eshell V7.3  (abort with ^G)
1> Point = {4,5}.
{4,5}
2> Tagged_point = {point, Point}.
{point,{4,5}}
3> {T,P}=Tagged_point .
{point,{4,5}}
4> T
4> .
point
5> P .
{4,5}
6>
```

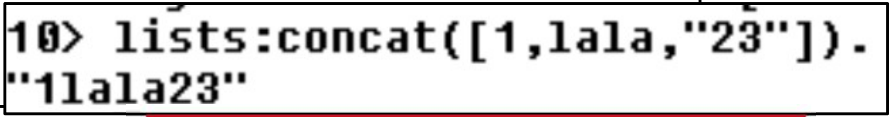


```
Eshell V7.3  (abort with ^G)
1> Point = {4,5}.
{4,5}
2> L = [1,Point].
[1,{4,5}]
3> Head = hd(L).
1
4> Tail = tl(L).
[{4,5}]
5> [Point] == Tail .
true
6> New = [3|[6|Tail]] .
[3,6,{4,5}]
```

Continutul modulului **lists**



```
5> lists:
all/2           any/2           append/1        append/2        concat/1
delete/2        droplast/1      dropwhile/2     duplicate/2     filter/2
filtermap/2     flatlength/1    flatmap/2       flatten/1       flatten/2
foldl/3         foldr/3         foreach/2       keydelete/3     keyfind/3
keymap/3        keymember/3     keymerge/3      keyreplace/4    keysearch/3
keysort/2       keystore/4      keytake/3       last/1          map/2
mapfoldl/3      mapfoldr/3      max/1           member/2        merge/1
merge/2         merge/3         merge3/3        min/1           module_info/0
module_info/1   nth/2           nthtail/2       partition/2     prefix/2
reverse/1       reverse/2       rkeymerge/3     rmerge/2        rmerge/3
rmerge3/3       rukeymerge/3    rumerge/2       rumerge/3       rumerge3/3
seq/2           seq/3           sort/1          sort/2          split/2
splitwith/2     sublist/2       sublist/3       subtract/2      suffix/2
sum/1           takewhile/2     ukeymerge/3     ukeysort/2      umerge/1
umerge/2        umerge/3        umerge3/3       unzip/1         unzip3/1
usort/1         usort/2         zf/2            zip/2           zip3/3
zipwith/3       zipwith3/4
5> lists:
```



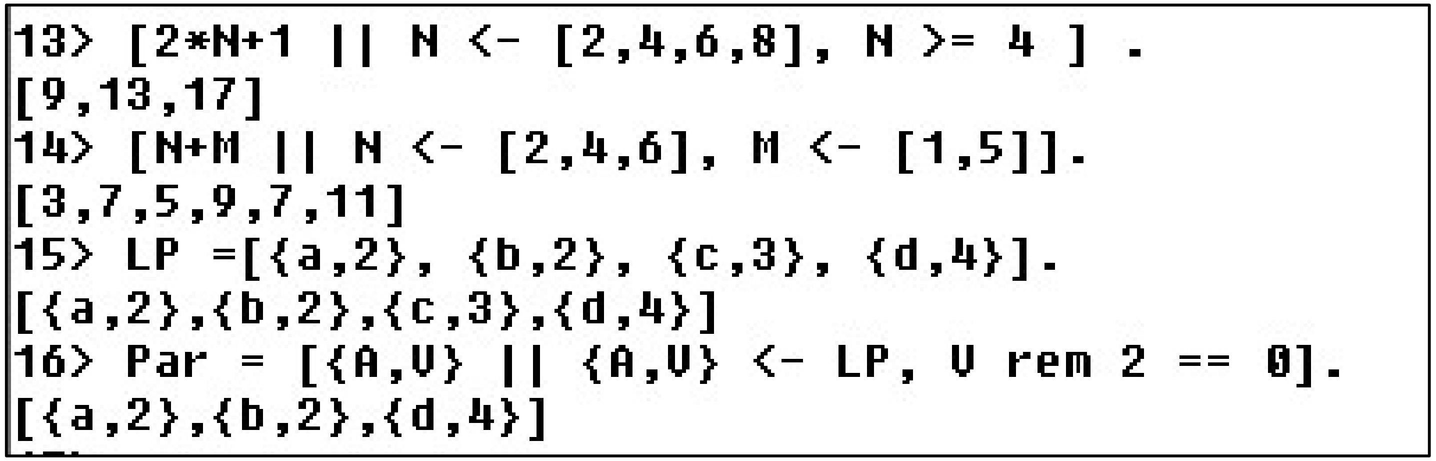
```
10> lists:concat([1,lala,"23"]).
"1lala23"
```

http://erlang.org/doc/man/lists.html

modul:functie(argumente).

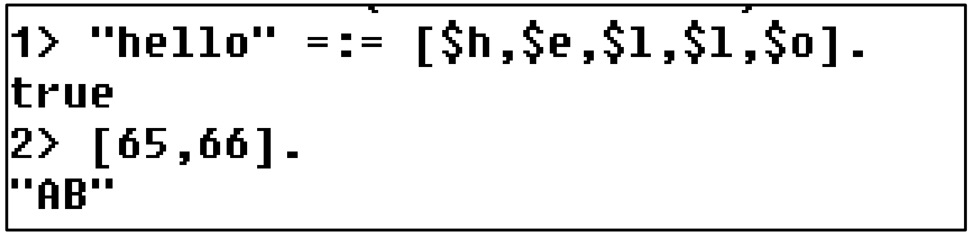
Tipuri de date

- Liste: definirea listelor prin comprehensiune



```
13> [2*N+1 || N <- [2,4,6,8], N >= 4 ] .
[9,13,17]
14> [N+M || N <- [2,4,6], M <- [1,5]].
[3,7,5,9,7,11]
15> LP =[{a,2}, {b,2}, {c,3}, {d,4}].
[{a,2},{b,2},{c,3},{d,4}]
16> Par = [{A,V} || {A,V} <- LP, V rem 2 == 0].
[{a,2},{b,2},{d,4}]
```

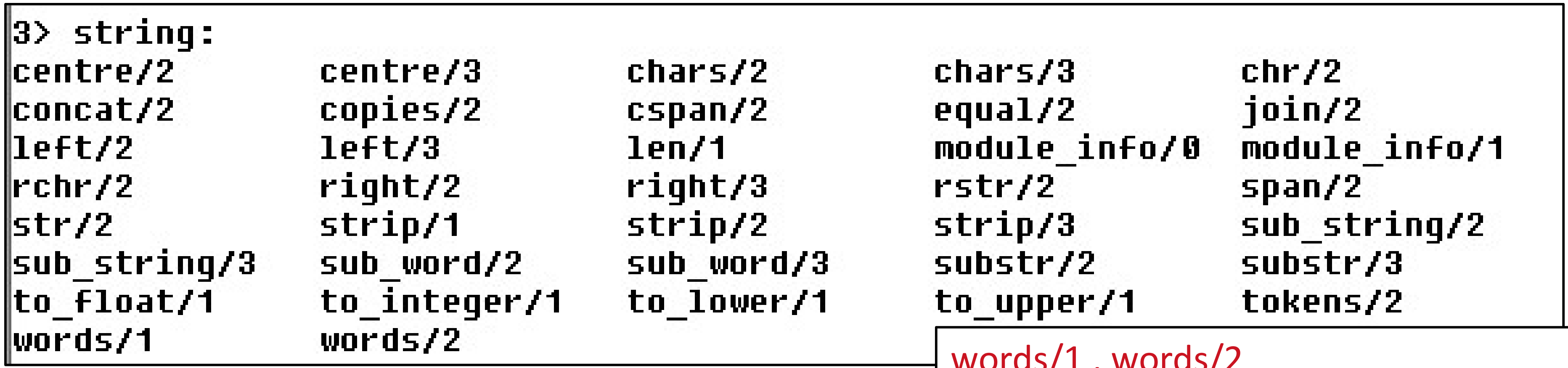
- String: "hello" (notatii pentru lista codurilor ASCII)



```
1> "hello" =:= [$h,$e,$l,$l,$o].
true
2> [65,66].
"AB"
```

http://erlang.org/doc/reference_manual/data_types.html

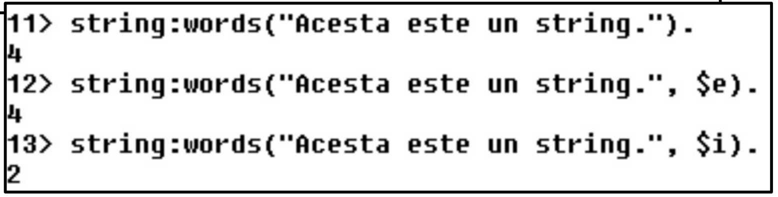
Continutul modulului **string**:



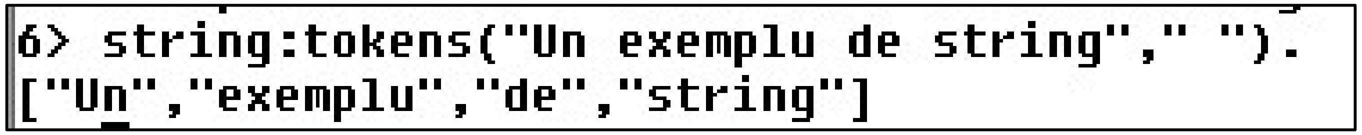
```
3> string:
centre/2          centre/3          chars/2           chars/3           chr/2
concat/2          copies/2          cspan/2           equal/2           join/2
left/2            left/3            len/1             module_info/0     module_info/1
rchr/2            right/2           right/3           rstr/2            span/2
str/2             strip/1           strip/2           strip/3           sub_string/2
sub_string/3      sub_word/2        sub_word/3        substr/2          substr/3
to_float/1        to_integer/1      to_lower/1        to_upper/1        tokens/2
words/1           words/2
```

http://erlang.org/doc/man/string.html

words/1 , words/2
doua functii diferite pot avea acelasi nume daca au un numar diferit de argumente



```
11> string:words("Acesta este un string.").
4
12> string:words("Acesta este un string.", $e).
4
13> string:words("Acesta este un string.", $i).
2
```



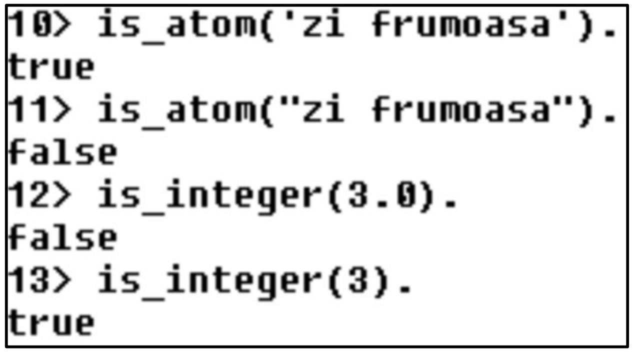
```
6> string:tokens("Un exemplu de string"," ").
["Un","exemplu","de","string"]
```

modul:functie(argumente).

Conversii explicite:

```
1> atom_to_list(hello).
"hello"
2> list_to_atom("hello").
Hello
3> float_to_list(7.0).
"7.00000000000000000000e+00"
4> list_to_float("7.000e+00").
7.0
5> integer_to_list(77).
"77"
6> list_to_integer("77").
77
7> tuple_to_list({a,b,c}).
[a,b,c]
8> list_to_tuple([a,b,c]).
{a,b,c}
```

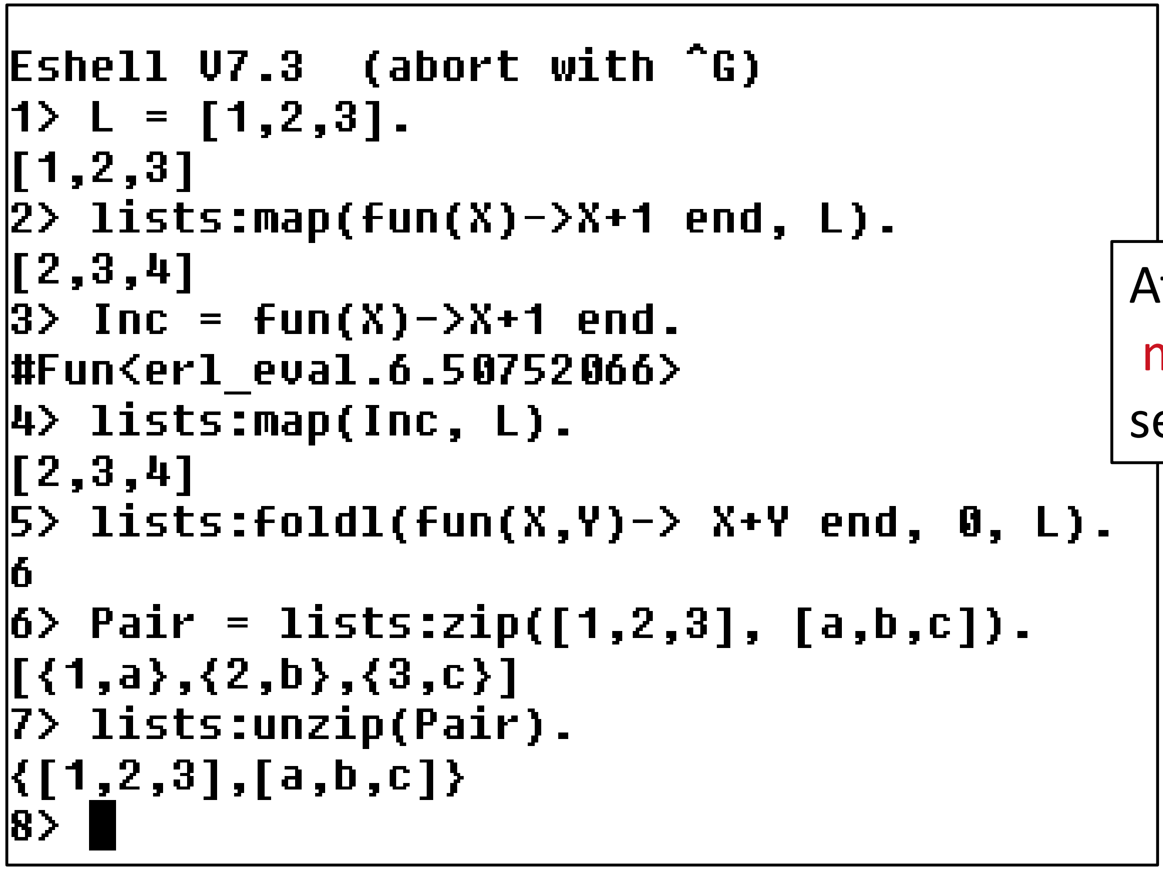
Type-tests:



```
10> is_atom('zi frumoasa').
true
11> is_atom("zi frumoasa").
false
12> is_integer(3.0).
false
13> is_integer(3).
true
```

http://erlang.org/doc/reference_manual/data_types.html

Functii de nivel inalt

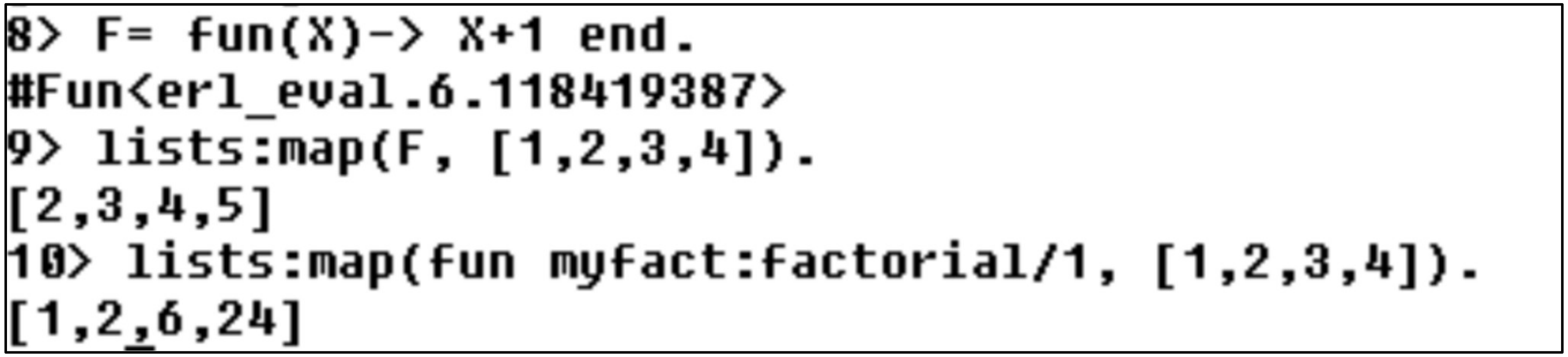


```
Eshell V7.3  (abort with ^G)
1> L = [1,2,3].
[1,2,3]
2> lists:map(fun(X)->X+1 end, L).
[2,3,4]
3> Inc = fun(X)->X+1 end.
#Fun<erl_eval.6.50752066>
4> lists:map(Inc, L).
[2,3,4]
5> lists:foldl(fun(X,Y)-> X+Y end, 0, L).
6
6> Pair = lists:zip([1,2,3], [a,b,c]).
[{1,a},{2,b},{3,c}]
7> lists:unzip(Pair).
{[1,2,3],[a,b,c]}
8> 
```

Atentie!
map, zip, foldl
se gasesc in modulul lists

http://erlang.org/doc/programming_examples/funs.html

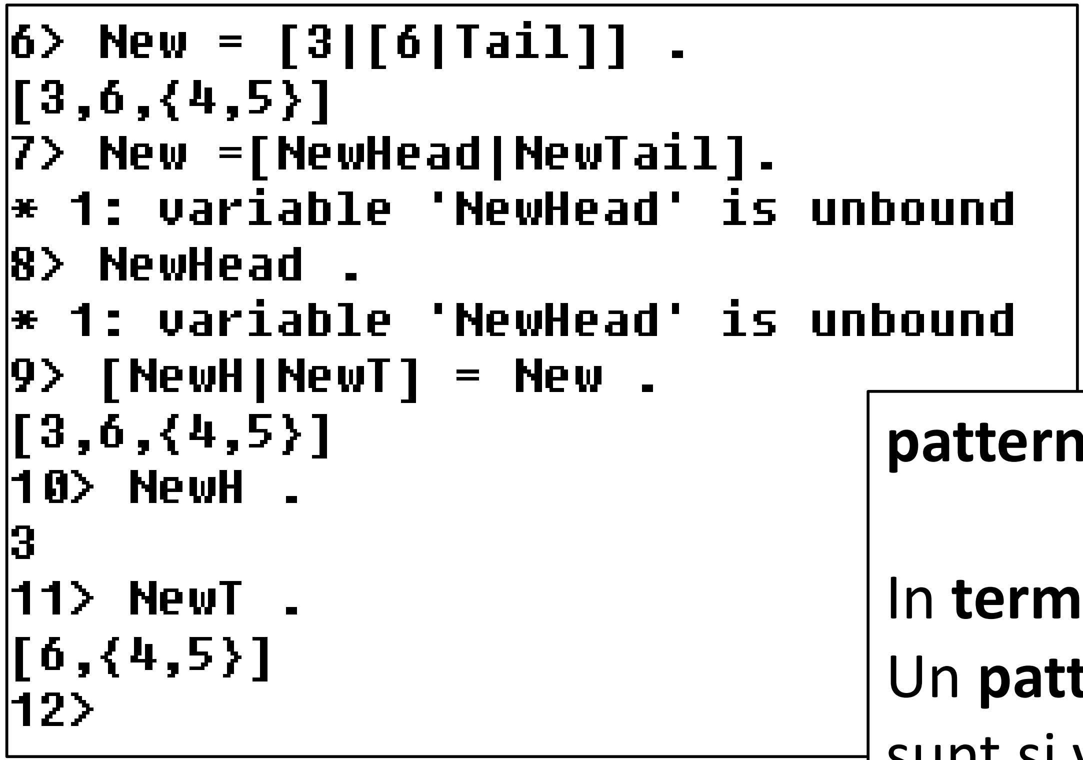
Functii de nivel inalt



```
8> F= fun(X)-> X+1 end.
#Fun<erl_eval.6.118419387>
9> lists:map(F, [1,2,3,4]).
[2,3,4,5]
10> lists:map(fun myfact:factorial/1, [1,2,3,4]).
[1,2,6,24]
```

http://erlang.org/doc/programming_examples/funs.html

Pattern matching



```
6> New = [3|[6|Tail]] .
[3,6,{4,5}]
7> New =[NewHead|NewTail].
* 1: variable 'NewHead' is unbound
8> NewHead .
* 1: variable 'NewHead' is unbound
9> [NewH|NewT] = New .
[3,6,{4,5}]
10> NewH .
3
11> NewT .
[6,{4,5}]
12>
```

pattern = termen

In **termen** toate variabilele sunt legate
Un **pattern** este ca un termen in care sunt si variabile libere

Module

```
-module(mymod).
-export([hello/2,factorial/1, start/0]).
```

atribute

```
hello(S,X) -> io:format("Hello ~s, factorialul este ~p!~n",[S,X]).

factorial(0) -> 1;
factorial(N) -> N * factorial(N-1).

start() ->
   {ok,[Name]}= io:fread("Your Name:", "~s"),
   {ok,[Val]}= io:fread("Your No:", "~d"),
   hello(Name, factorial(Val)).
```

declaratii de functii

http://erlang.org/doc/reference_manual/modules.html

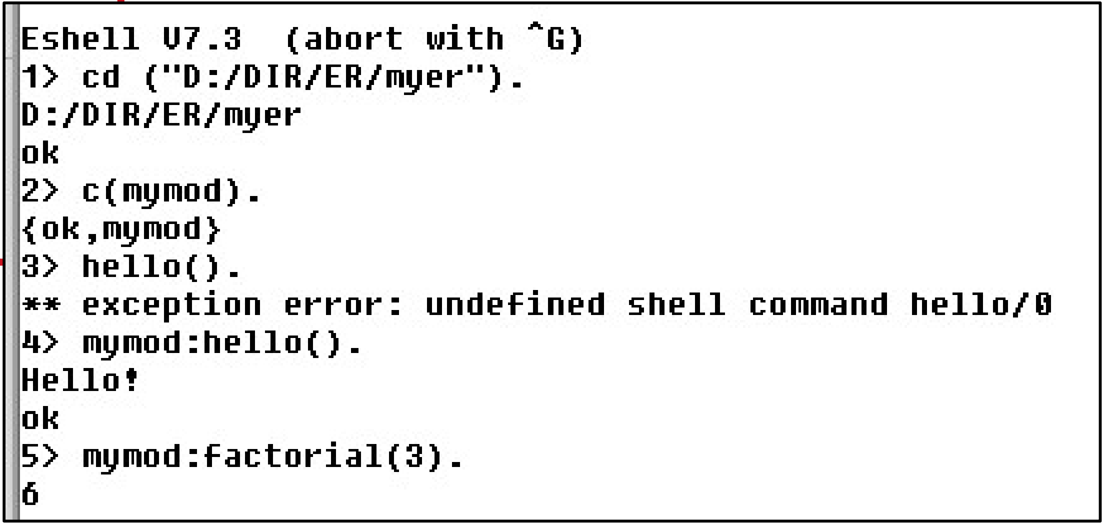
Module

mymod.erl

numele fisierului coincide cu numele modulului

```
  -module(mymod).                    %attribute
  -export([hello/0,factorial/1]).     %attribute
% -compile(export_all).

hello() -> io:format("Hello!~n").    %function

factorial(0) -> 1;                     %function
factorial(N) -> N * factorial(N-1).
```

modul:functie(argumente)
o functie e unic determinate de
(modul, nume, aritate)

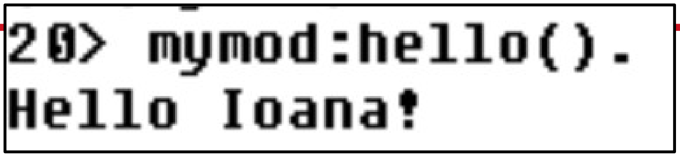


```
Eshell V7.3  (abort with ^G)
1> cd ("D:/DIR/ER/myer").
D:/DIR/ER/myer
ok
2> c(mymod).
{ok,mymod}
3> hello().
** exception error: undefined shell command hello/0
4> mymod:hello().
Hello!
ok
5> mymod:factorial(3).
6
```

Module

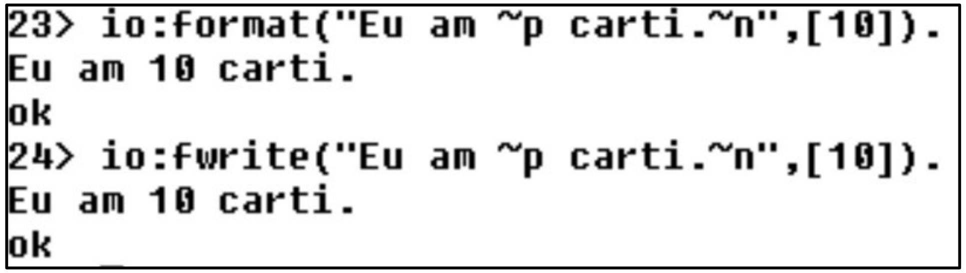
mymod.erl

```
  -module(mymod).                    %attribute
  -export([hello/0,factorial/1]).     %attribute
  -define(Eu, "Ioana")                %macros

hello() -> io:format("Hello, ~s!~n",[?Eu]).    %function
```



```
20> mymod:hello().
Hello Ioana!
```

io:format/io:fwrite



```
23> io:format("Eu am ~p carti.~n",[10]).
Eu am 10 carti.
ok
24> io:fwrite("Eu am ~p carti.~n",[10]).
Eu am 10 carti.
ok
```

erlang.org/doc/man/io.html

Definirea functiilor
se face folosind pattern-uri

```
prels("a"++ L) -> io:format("~s ~n",[L++L]);        clauza

prels("b"++ L) -> io:format("~s ~n",[L++"b"]);        clauza

prels(_) ->  io:format("Nu incepe cu \"a\" sau \"b\". ~n").        clauza
```

Definirea functiilor

O declaratie de functie este o secventa de clauze separate prin ; care se termina cu .

```
Name(Pattern11,...,Pattern1N) [when GuardSeq1] ->
   Body1;
...;
Name(PatternK1,...,PatternKN) [when GuardSeqK] ->
   BodyK.
```

Body

```
Expr1,
...,
ExprN
```

Corpul unei clause este o secventa de expresii separate prin ,

http://erlang.org/doc/reference_manual/functions.html

Definirea functiilor folosind garzi (**when**)

```
par(X) -> (X rem 2 == 0) .
preln(X) when par(X) -> io:format("Este par ~n");  %gresit
```

nu se accepta functii definite de utilizator in garzi

Corect!

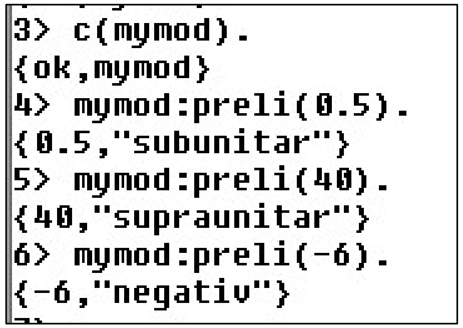
```
prelg(X) when (X rem 2 == 0) -> io:format("Este par ~n");
prelg(_) ->    io:format("Este impar ~n").
```

Definirea functiilor

if .. end

```
preli(X) ->
    Rez = if  ((X =< 1) and (X >= 0)) -> "subunitar";
                                  (X > 1) -> "supraunitar";
                                     true -> "negativ"
          end,
   {X,Rez}.
```

obligatorie

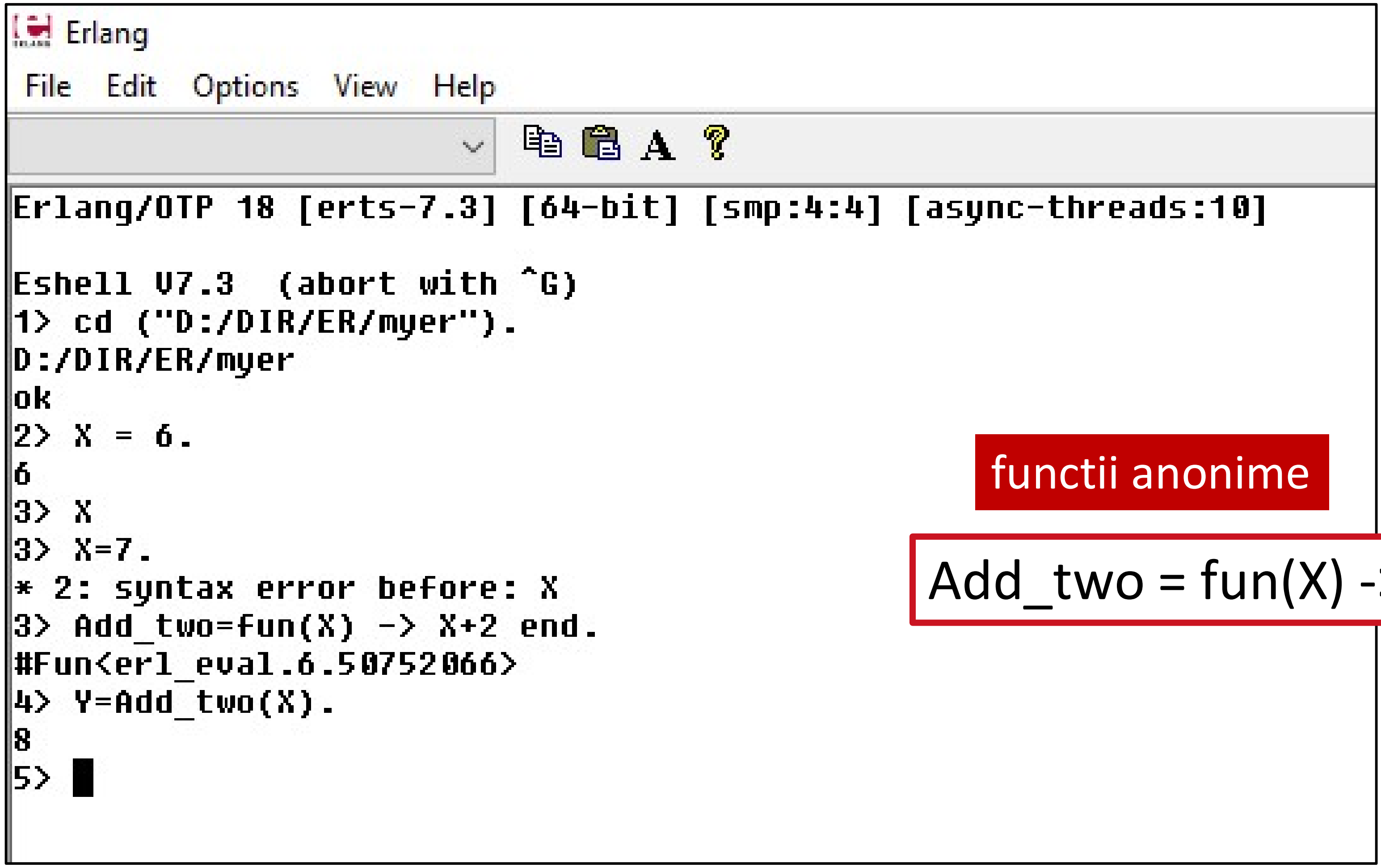


```
3> c(mymod).
{ok,mymod}
4> mymod:preli(0.5).
{0.5,"subunitar"}
5> mymod:preli(40).
{40,"supraunitar"}
6> mymod:preli(-6).
{-6,"negativ"}
```

case .. end

```
prelc({S,X}) ->  case {S,X} of
                   {"pozitiv", X} when ((X =< 1) and (X >= 0)) -> "subunitar";
                   {"pozitiv",X} when (X>1) -> "supraunitar";
                   {_,X} when (X >= 0) -> "pozitiv";
                                     _ -> "negativ"
                 end.
```

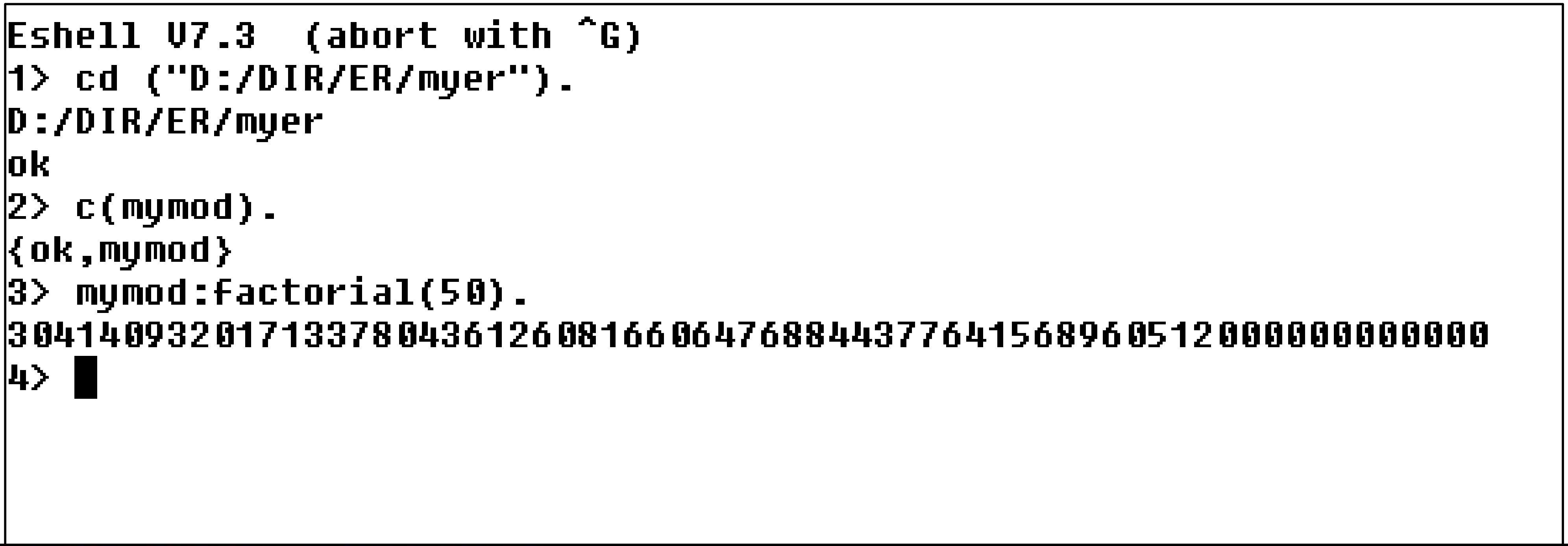
Definirea functiilor



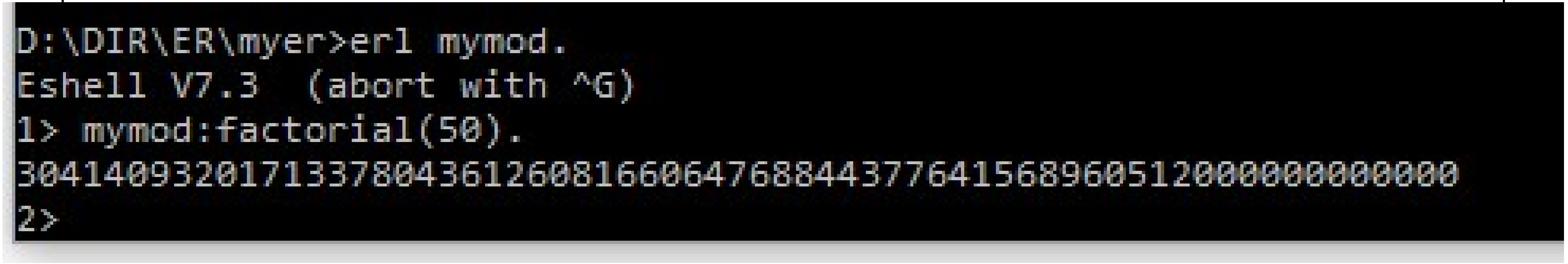
```
Erlang/OTP 18 [erts-7.3] [64-bit] [smp:4:4] [async-threads:10]

Eshell V7.3  (abort with ^G)
1> cd ("D:/DIR/ER/myer").
D:/DIR/ER/myer
ok
2> X = 6.
6
3> X
3> X=7.
* 2: syntax error before: X
3> Add_two=fun(X) -> X+2 end.
#Fun<erl_eval.6.50752066>
4> Y=Add_two(X).
8
5> 
```

functii anonime

Add_two = fun(X) -> X+2 end.



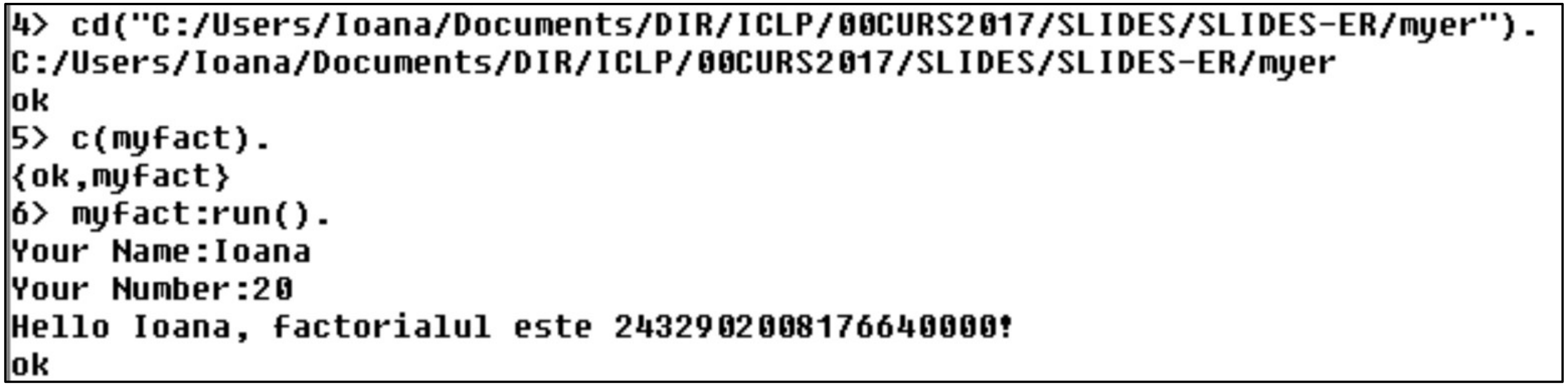
```
Eshell V7.3  (abort with ^G)
1> cd ("D:/DIR/ER/myer").
D:/DIR/ER/myer
ok
2> c(mymod).
{ok,mymod}
3> mymod:factorial(50).
30414093201713378043612608166064768844377641568960512000000000000
4> 
```



```
D:\DIR\ER\myer>erl mymod.
Eshell V7.3  (abort with ^G)
1> mymod:factorial(50).
30414093201713378043612608166064768844377641568960512000000000000
2>
```

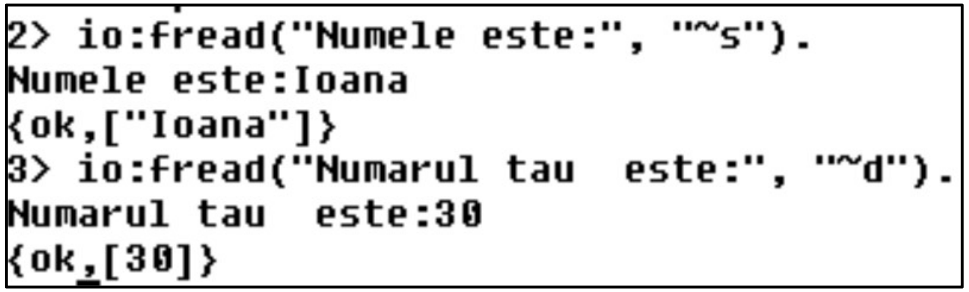
myfact.erl

```
-module(myfact).
-export([run/0]).

factorial(0) -> 1;
factorial(N) -> N * factorial(N-1).

 hello(S,X) -> io:format("Hello ~s, factorialul este ~p!~n",[S,X]).

run() ->
      {ok,[Name]}= io:fread("Your Name:", "~s"),
      {ok,[Val]}= io:fread("Your Number:", "~d"),
      hello(Name, factorial(Val)).
```



```
4> cd("C:/Users/Ioana/Documents/DIR/ICLP/00CURS2017/SLIDES/SLIDES-ER/myer").
C:/Users/Ioana/Documents/DIR/ICLP/00CURS2017/SLIDES/SLIDES-ER/myer
ok
5> c(myfact).
{ok,myfact}
6> myfact:run().
Your Name:Ioana
Your Number:20
Hello Ioana, factorialul este 2432902008176640000!
ok
```

io:fread



```
2> io:fread("Numele este:", "~s").
Numele este:Ioana
{ok,["Ioana"]}
3> io:fread("Numarul tau  este:", "~d").
Numarul tau  este:30
{ok,[30]}
```

erlang.org/doc/man/io.html

Erlang-99: 99 Erlang Problems
https://purijatin.github.io/newsletters/erlang-99/

Concurenta in Erlang

Jim Larson, Erlang for Concurrent Programming, ACM Queue, 2008

CONCURRENCY IN ERLANG

lightweight processes with asynchronous message passing

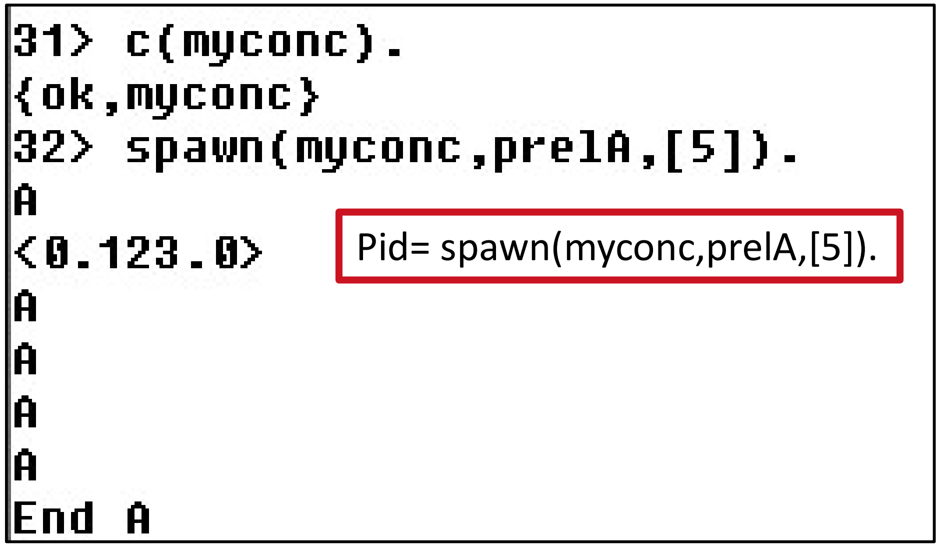
Procesele in Erlang:

- pot fi create si distruse rapid
- comunica prin mesaje, iar comunicarea este rapida
- sunt complet independente din punctul de vedere al memoriei

➢ Crearea proceselor: spawn

Functia spawn creaza un process care este executat in parallel cu procesul care l-a creat si intoarce un **Pid** (Process Identifier) , care este folosit pentru trimiterea mesajelor.

spawn/3
spawn(modul, functie, lista argumentelor)
Pid = spawn(modul, functie, lista argumentelor)

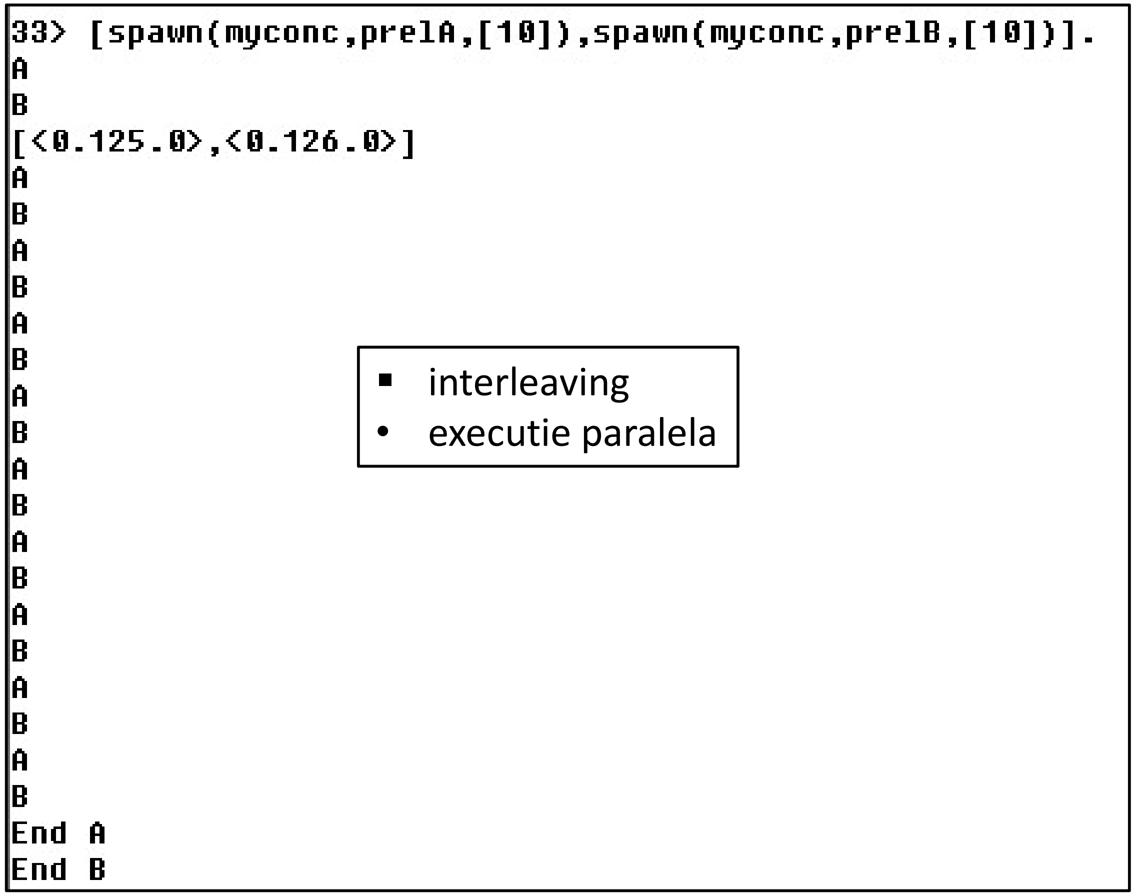


```
31> c(myconc).
{ok,myconc}
32> spawn(myconc,prelA,[5]).
A
<0.123.0>
A
A
A
A
End A
```

```
Pid= spawn(myconc,prelA,[5]).
```

```
-module(myconc).
-export([prelA/1).

prelA(X) when (X == 0) -> io:format("End A ~n");
prelA(X) when (X > 0) -> io:format("A ~n"), prelA(X-1);
prelA(_) ->       io:format("error ~n").
```

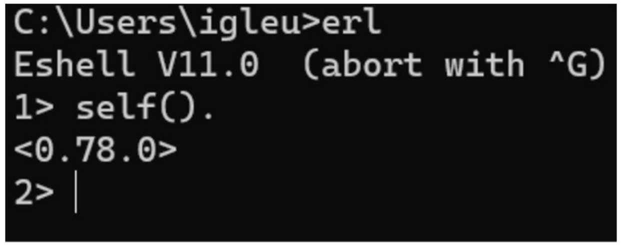
Exemplu: doua procese care sunt executate in paralel



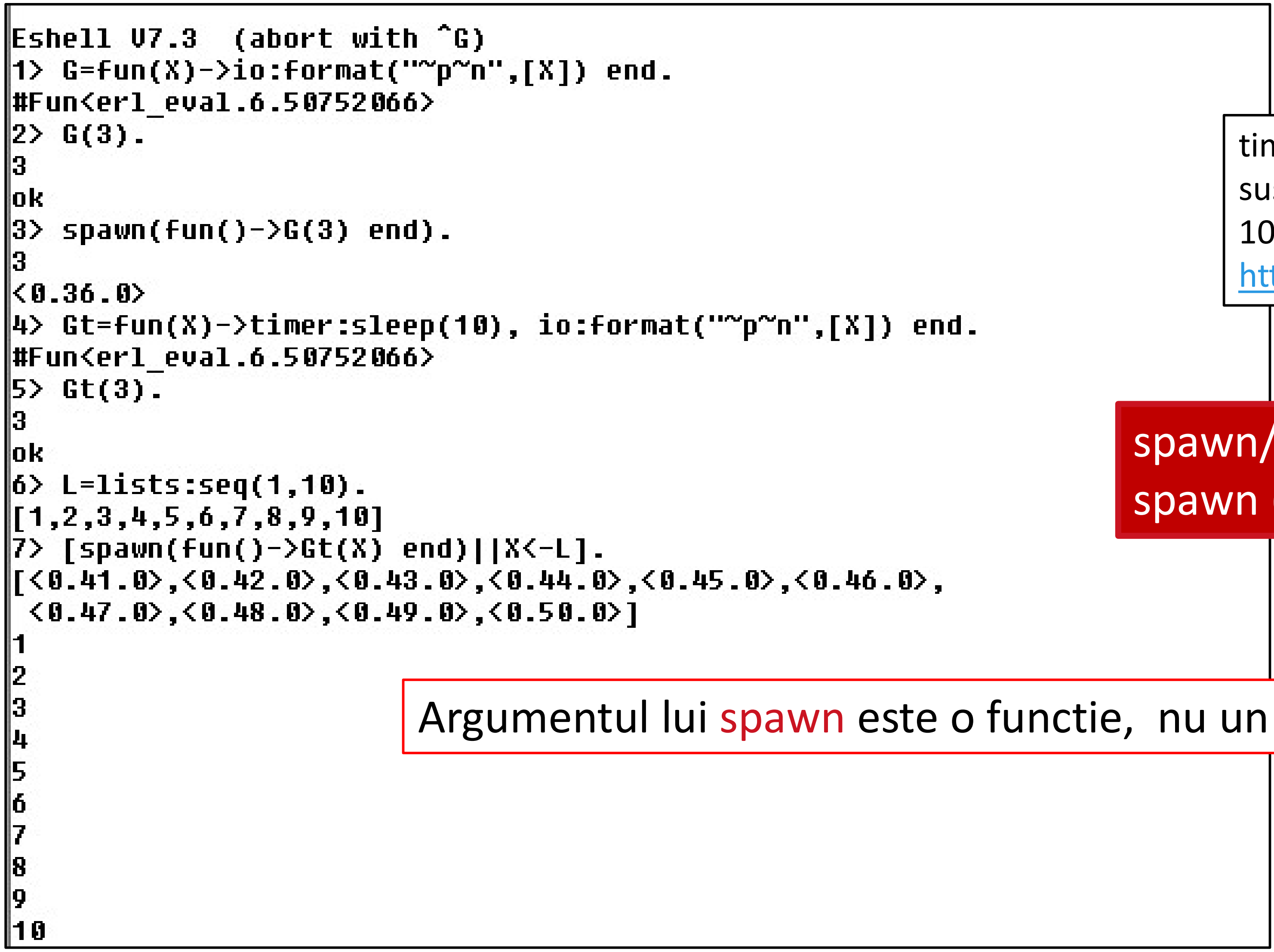
```
33> [spawn(myconc,prelA,[10]),spawn(myconc,prelB,[10])].
A
B
[<0.125.0>,<0.126.0>]
A
B
A
B
A
B
A
B
A
B
A
B
A
B
A
B
A
B
End A
End B
```

- interleaving
- executie paralela

- Un proces este identificat printr-un **"process identifier (pid)".**
- Un **pid** este un tip de date in Erlang https://www.erlang.org/doc/reference_manual/data_types.html#pid
- In interiorul unui proces, functia self() intoarce pid-ul procesului.
- In Erlang, shell-ul este un proces.



```
C:\Users\igleu>erl
Eshell V11.0  (abort with ^G)
1> self().
<0.78.0>
2> |
```



```
Eshell V7.3  (abort with ^G)
1> G=fun(X)->io:format("~p~n",[X]) end.
#Fun<erl_eval.6.50752066>
2> G(3).
3
ok
3> spawn(fun()->G(3) end).
3
<0.36.0>
4> Gt=fun(X)->timer:sleep(10), io:format("~p~n",[X]) end.
#Fun<erl_eval.6.50752066>
5> Gt(3).
3
ok
6> L=lists:seq(1,10).
[1,2,3,4,5,6,7,8,9,10]
7> [spawn(fun()->Gt(X) end)||X<-L].
[<0.41.0>,<0.42.0>,<0.43.0>,<0.44.0>,<0.45.0>,<0.46.0>,
 <0.47.0>,<0.48.0>,<0.49.0>,<0.50.0>]
1
2
3
4
5
6
7
8
9
10
```

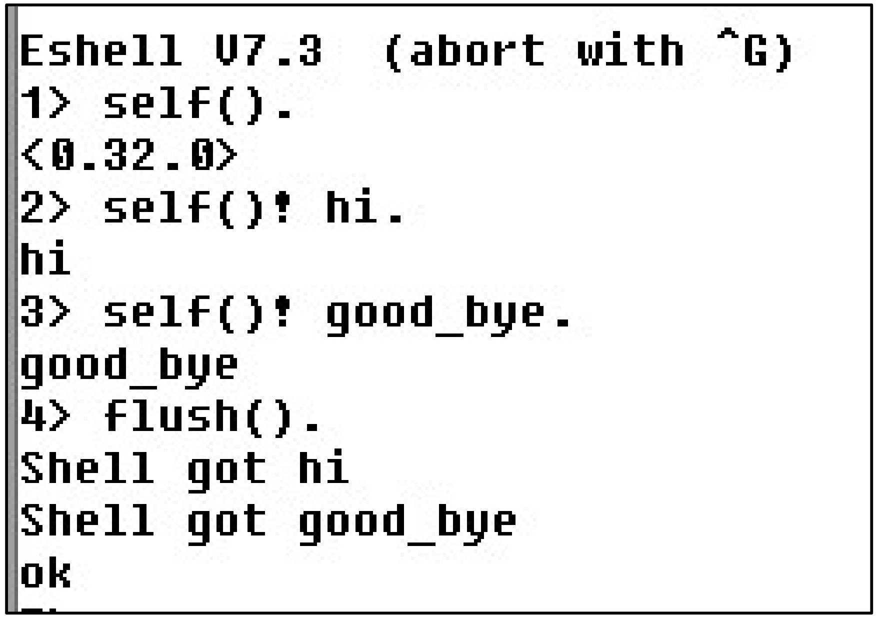
timer:sleep(10)
suspenda procesul pentru
10 milisecunde
http://erlang.org/doc/man/timer.html

spawn/1
spawn (fun() -> Gt(X) end)

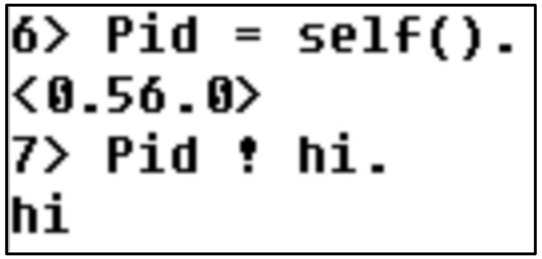
Argumentul lui spawn este o functie, nu un apel de functie.

➢Trimiterea mesajelor: **Pid ! msg**

Mesajul **msg** este trimis procesului cu id-ul **Pid.** Mesajul este un termen Erlang.



```
Eshell V7.3  (abort with ^G)
1> self().
<0.32.0>
2> self()! hi.
hi
3> self()! good_bye.
good_bye
4> flush().
Shell got hi
Shell got good_bye
ok
```



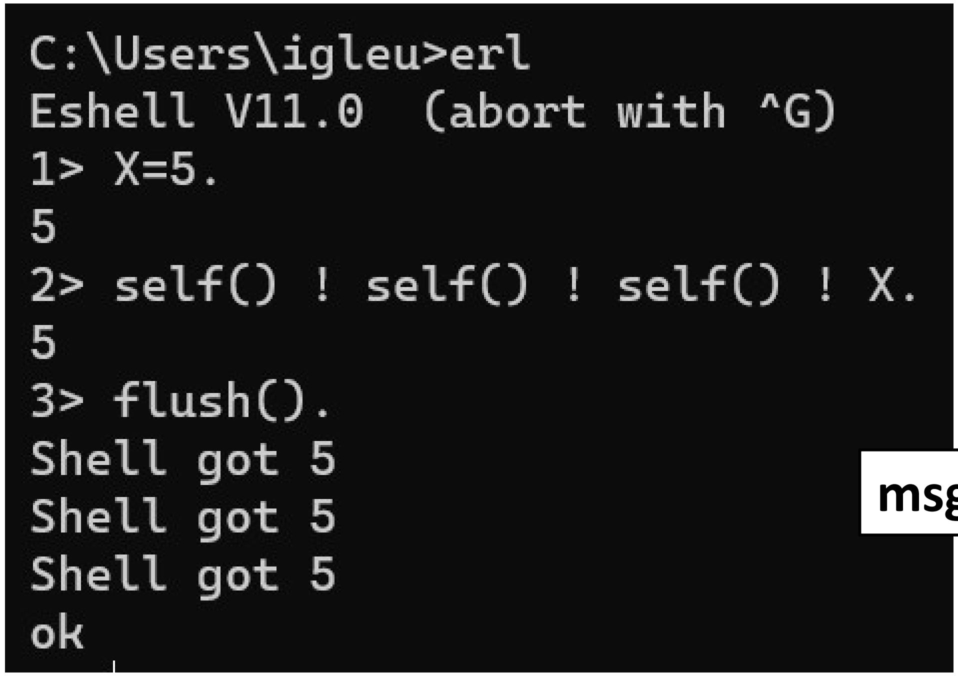
```
6> Pid = self().
<0.56.0>
7> Pid ! hi.
hi
```

flush()
elimina mesajele trimise shell-ului

➢Trimiterea mesajelor: **Pid ! msg**

Mesajul **msg** este trimis procesului cu id-ul **Pid.** Mesajul este un termen Erlang.

Pid1 ! Pid2 ! ! Pidn ! msg

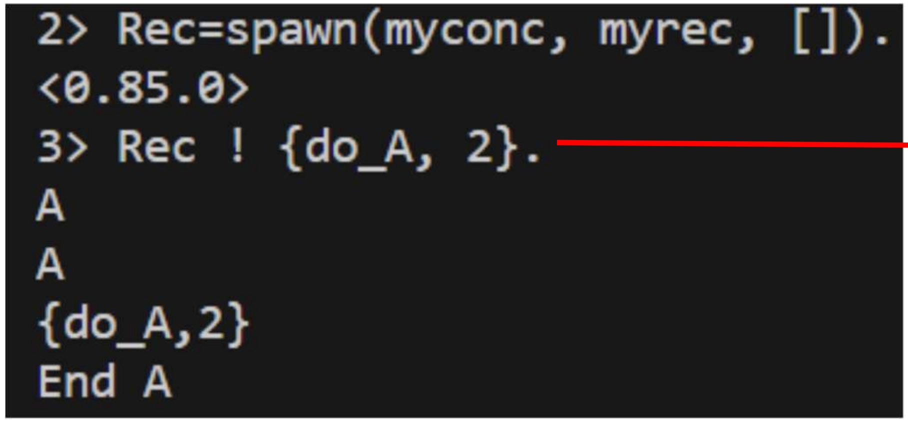


```
C:\Users\igleu>erl
Eshell V11.0  (abort with ^G)
1> X=5.
5
2> self() ! self() ! self() ! X.
5
3> flush().
Shell got 5
Shell got 5
Shell got 5
ok
```

msg este evaluat

➢ Primirea mesajelor

Mesajul **msg** este trimis procesului cu id-ul **Pid.** Mesajul este un termen Erlang.



```
2> Rec=spawn(myconc, myrec, []).
<0.85.0>
3> Rec ! {do_A, 2}.
A
A
{do_A,2}
End A
```

procesul Rec primeste mesajul {do_A, 2}

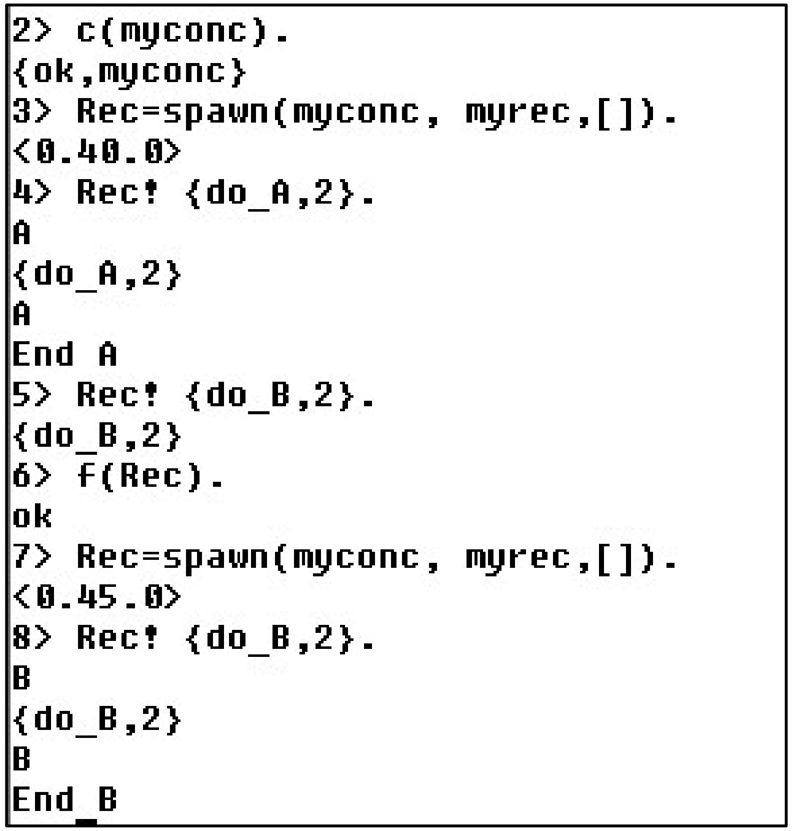
raspunsul este definit
in instructiunea
receive … end

```
myrec() ->
  receive
   {do_A, X} ->  prelA(X);
   {do_B, X} ->  prelB(X);
             _ -> io:format("Nothing to do ~n")
  end.
```

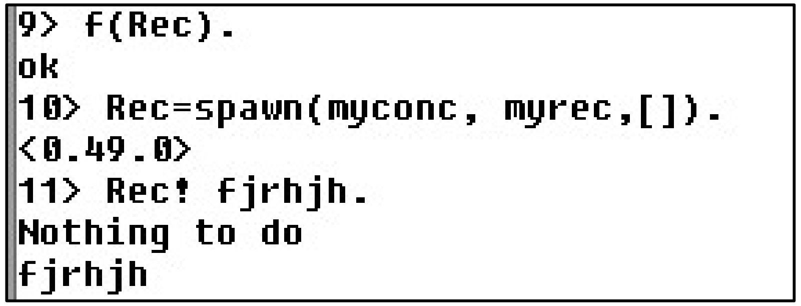
➢ Primirea mesajelor

Mesajul **msg** este trimis procesului cu id-ul **Pid.** Mesajul este un termen Erlang.

```
myrec() ->
  receive
   {do_A, X} ->  prelA(X);
   {do_B, X} ->  prelB(X);
              _ -> io:format("Nothing to do ~n")
  end.
```



```
2> c(myconc).
{ok,myconc}
3> Rec=spawn(myconc, myrec,[]).
<0.40.0>
4> Rec! {do_A,2}.
A
{do_A,2}
A
End A
5> Rec! {do_B,2}.
{do_B,2}
6> f(Rec).
ok
7> Rec=spawn(myconc, myrec,[]).
<0.45.0>
8> Rec! {do_B,2}.
B
{do_B,2}
B
End_B
```



```
9> f(Rec).
ok
10> Rec=spawn(myconc, myrec,[]).
<0.49.0>
11> Rec! fjrhjh.
Nothing to do
fjrhjh
```

➢ receive … end

```
receive
Pattern1 when Guard1 -> Expr1;
Pattern2 when Guard2 -> Expr2;
Pattern3 -> Expr3
end
```

- Cand ajunge la o instructiune **receive** un proces scoate un mesaj din coada de mesaje si incearca sa ii gaseasca un sablon.

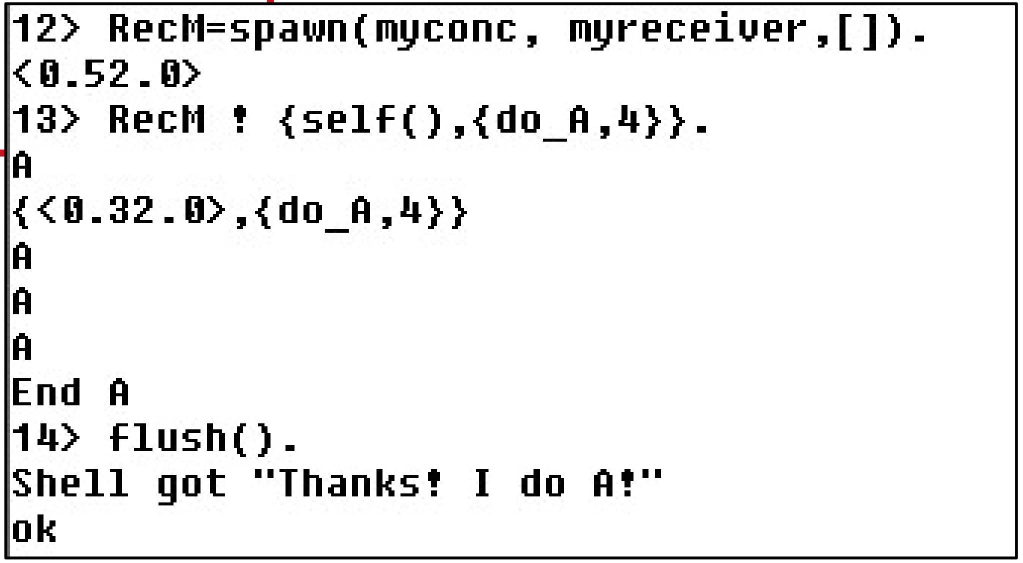
- Daca coada de mesaje este vida procesul se blocheaza si asteapta un mesaj care se potriveste cu un sablon.

- trimiterea mesajelor se face asincron
- receive este singura instructiune care blocheaza procesul

➢ Schimb de mesaje intre procese

```
myreceiver() ->
  receive
  {From, {do_A, X}} -> From ! "Thanks! I do A!",
                                            prelA(X);
  {From, {do_B, X}} -> From! "Thanks! I do B!",
                                            prelB(X);
        _    -> io:format("Nothing to do ~n")
  end.
```

schimb de mesaje intre **Rec** si **shell**



```
12> RecM=spawn(myconc, myreceiver,[]).
<0.52.0>
13> RecM ! {self(),{do_A,4}}.
A
{<0.32.0>,{do_A,4}}
A
A
A
End A
14> flush().
Shell got "Thanks! I do A!"
ok
```

"Messages between Erlang processes are simply valid Erlang terms. That is, they can be lists, tuples, integers, atoms, pids, and so on.

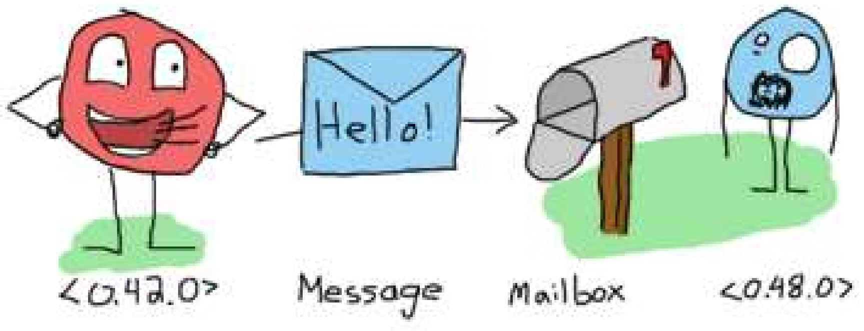
Each process has its own input queue for messages it receives. New messages received are put at the end of the queue. When a process executes a receive, the first message in the queue is matched against the first pattern in the receive. If this matches, the message is removed from the queue and the actions corresponding to the pattern are executed.

However, if the first pattern does not match, the second pattern is tested. If this matches, the message is removed from the queue and the actions corresponding to the second pattern are executed. If the second pattern does not match, the third is tried and so on until there are no more patterns to test. If there are no more patterns to test, the first message is kept in the queue and the second message is tried instead. If this matches any pattern, the appropriate actions are executed and the second message is removed from the queue (keeping the first message and any other messages in the queue). If the second message does not match, the third message is tried, and so on, until the end of the queue is reached. If the end of the queue is reached, the process blocks (stops execution) and waits until a new message is received and this procedure is repeated."

http://erlang.org/doc/getting_started/conc_prog.html

- Transmiterea mesajelor este asincrona.

Datorita cozii pentru mesaje,
procesul care transmite mesajul
nu asteapta o confirmare de primire sau
prelucrarea acestuia,
mesajul intra in coada si asteapta
pana cand va fi procesat



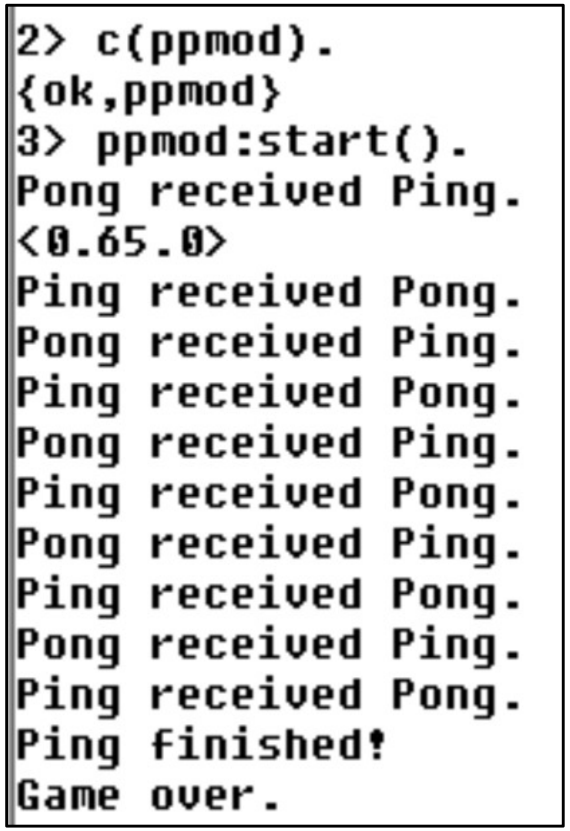
http://learnyousomeerlang.com/the-hitchhikers-guide-to-concurrency#dont-panic

➢ Concurenta in Erlang este implementata folosind urmatoarele primitive:

```
Pid = spawn (fun)

Pid = spawn (module, fct, args)

Pid ! Message

receive … end
```

https://www.erlang.org/doc/man/erlang.html#spawn-4

➢ Ping - Pong



```
2> c(ppmod).
{ok,ppmod}
3> ppmod:start().
Pong received Ping.
<0.65.0>
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Ping finished!
Game over.
```

- Exista doua procese: Ping si Pong
- Procesul Ping trimite mesajul "Ping" si
  asteapta sa primeasca mesajul "Pong"
- Procesul Pong asteapta sa primeasca mesajul "Ping" si
  trimite mesajul "Pong"
- Procesul Pong este creat primul

http://erlang.org/doc/getting_started/conc_prog.html

➢ Ping - Pong

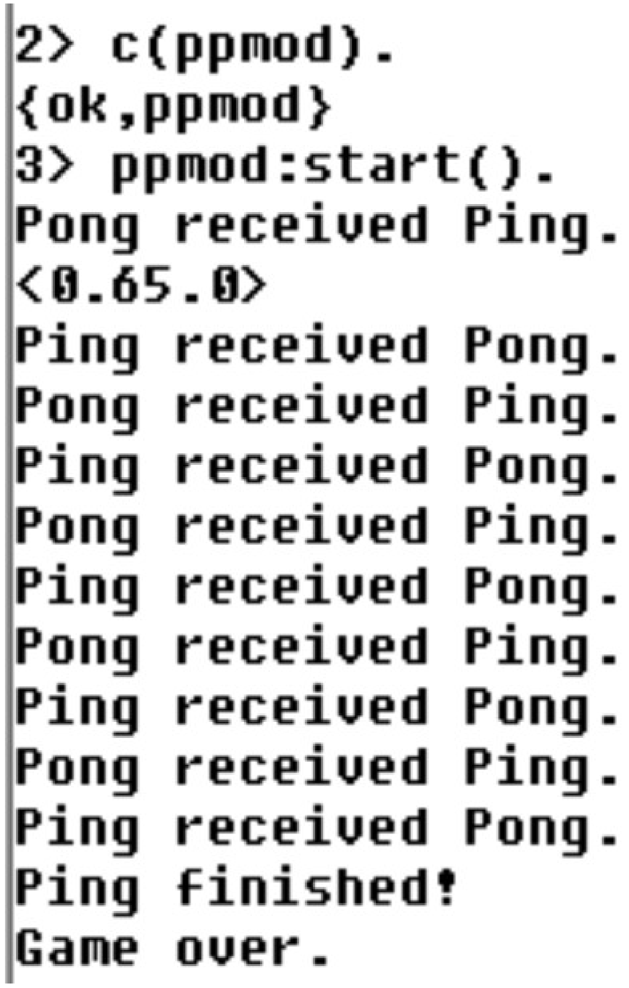
ppmod.erl

```
-module(ppmod).
-export([start/0,pingN/2,pong/0]).

pingN(Pid,0) -> Pid ! {self(), finished},
                io:format("Ping finished!~n");

pingN(Pid, N) -> Pid ! {self(),ping},
          receive
            {Pid, pong} -> io:format("Ping received Pong. ~n")
          end,
          pingN(Pid,N-1).
```

```
pong() ->
   receive
      {_,finished} -> io:format("Game over. ~n");
     {Pid, ping} -> io:format("Pong received Ping. ~n"),
                    Pid ! {self(),pong},
                     pong()
    end.
```

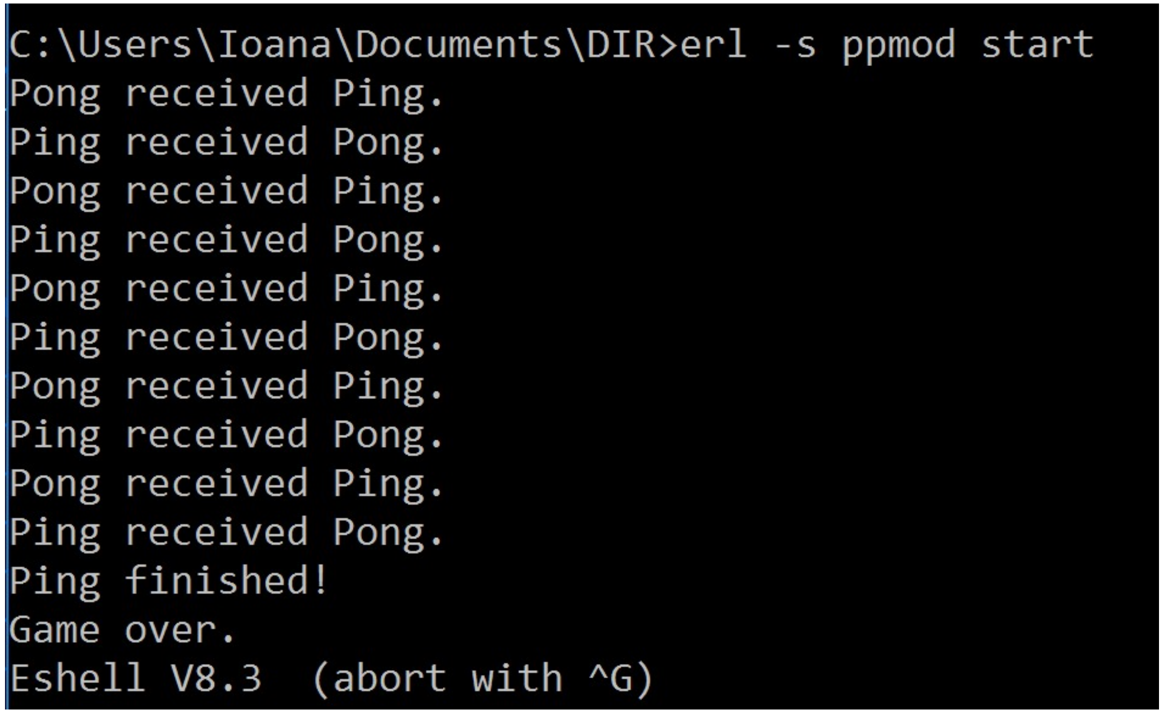
```
start() -> PongId = spawn(ppmod, pong,[]),
                   spawn(ppmod,pingN,[PongId,5]).
```

http://erlang.org/doc/getting_started/conc_prog.html



```
2> c(ppmod).
{ok,ppmod}
3> ppmod:start().
Pong received Ping.
<0.65.0>
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Ping finished!
Game over.
```

```
> erl  –s ppmod start
```



```
C:\Users\Ioana\Documents\DIR>erl -s ppmod start
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Pong received Ping.
Ping received Pong.
Ping finished!
Game over.
Eshell V8.3  (abort with ^G)
```

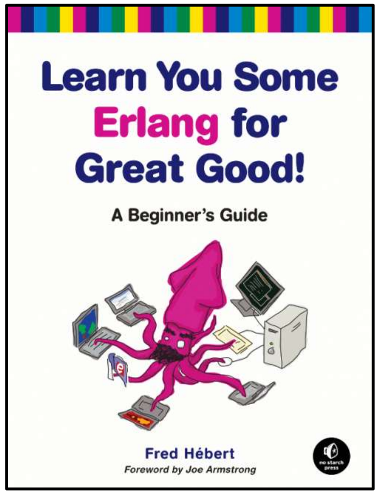
ACTOR MODEL

"Erlang's actor model can be imagined as a world where everyone is sitting alone in their own room and can perform a few distinct tasks. Everyone communicates strictly by writing letters and that's it. While it sounds like a boring life (and a new age for the postal service), it means you can ask many people to perform very specific tasks for you, and none of them will ever do something wrong or make mistakes which will have repercussions on the work of others; they may not even know the existence of people other than you (and that's great).

To escape this analogy, Erlang forces you to write actors (processes) that will share no information with other bits of code unless they pass messages to each other. Every communication is explicit, traceable and safe."

Fred Hébert, Learn You Some Erlang For Great Good

http://learnyousomeerlang.com/introduction#what-is-erlang

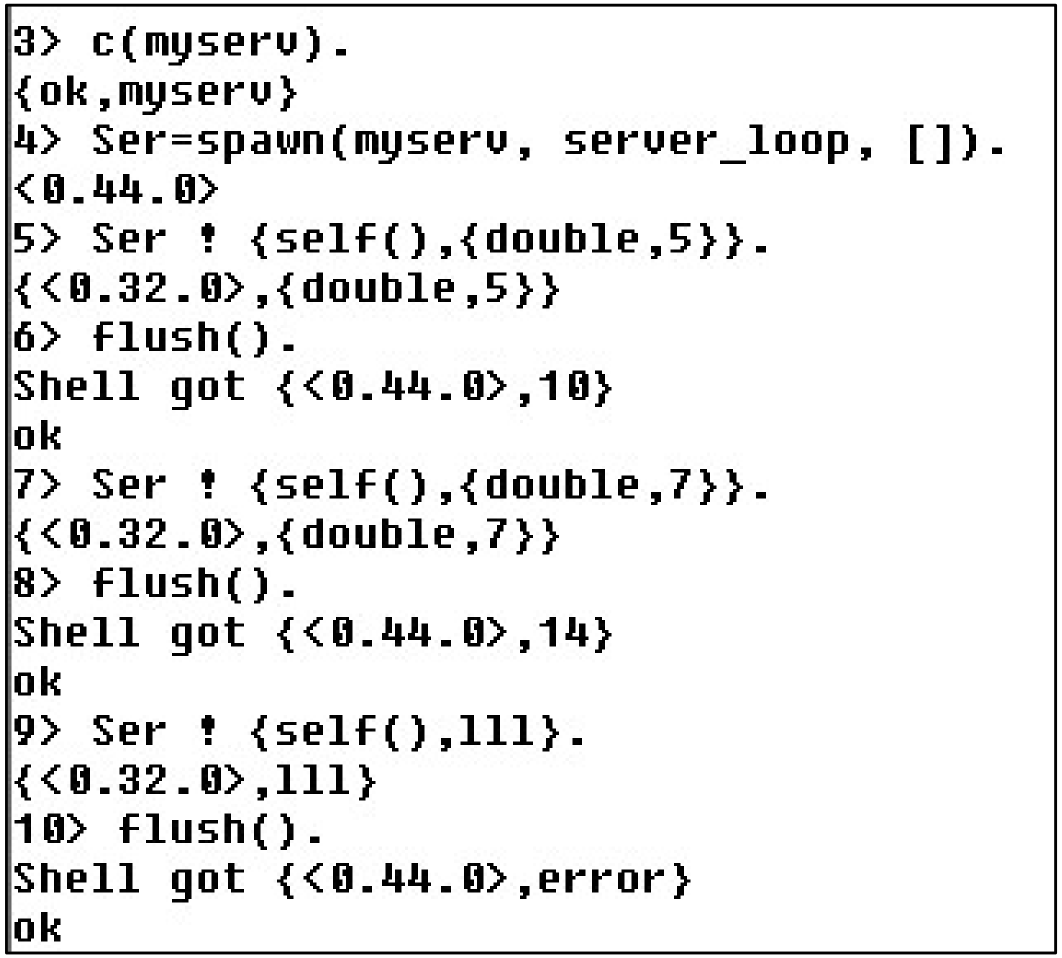

Learn You Some Erlang for Great Good!
A Beginner's Guide
Fred Hébert
Foreword by Joe Armstrong

Varianta online

➢Modelul Actori

- Introdus de Carl Hewitt in 1973
- Actorii sunt o notiune abstracta (corespunzatoare proceselor)
- Actorii au memorie proprie, NU au memorie partajata
- Actorii comunica prin mesaje
- Un actor este capabil sa:
    - trimita mesaje actorilor pe care ii cunoaste
    - creeze noi actori
    - raspunda mesajelor pe care le primeste
- Mesajele contin un destinatar si un continut
- Trimiterea mesajelor este asincrona

➢ Cilent-Server (Exemplu simplu: doubling service)

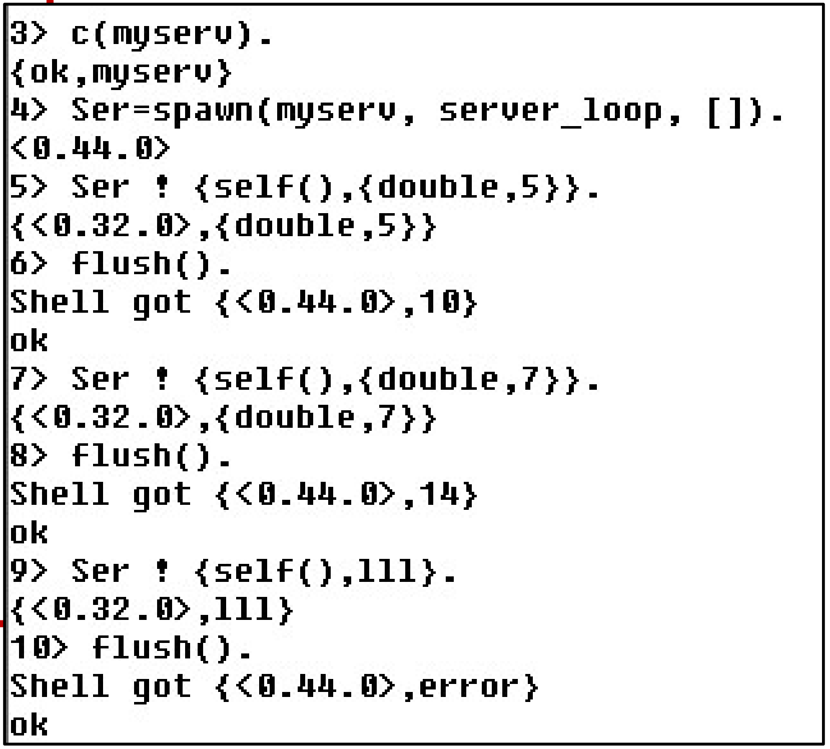


```
3> c(myserv).
{ok,myserv}
4> Ser=spawn(myserv, server_loop, []).
<0.44.0>
5> Ser ! {self(),{double,5}}.
{<0.32.0>,{double,5}}
6> flush().
Shell got {<0.44.0>,10}
ok
7> Ser ! {self(),{double,7}}.
{<0.32.0>,{double,7}}
8> flush().
Shell got {<0.44.0>,14}
ok
9> Ser ! {self(),111}.
{<0.32.0>,111}
10> flush().
Shell got {<0.44.0>,error}
ok
```

- Procesul **Ser** este serverul si executa functia **server_loop**
- Serverul primeste mesaje de la procese client si executa o actiune (dubleaza valoarea primita)
- In acest exemplu singurul client este shell-ul
- Mesajele primite de shell, adica raspunsurile trimise de server, sunt vizualizate cu **flush()**

➢ Cilent-Server (Exemplu simplu: doubling service)

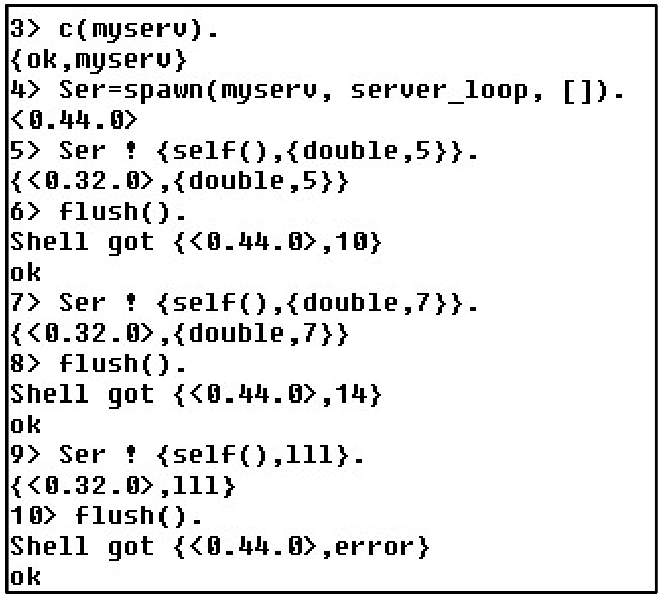
```
-module(myserv).
-export([server_loop/0]).

server_loop() ->
  receive
     {From, {double, Number}} -> From ! {self(),Number*2},
                                 server_loop() ;


        {From,_} -> From ! {self(),error},
                    server_loop()
  end.
```



```
3> c(myserv).
{ok,myserv}
4> Ser=spawn(myserv, server_loop, []).
<0.44.0>
5> Ser ! {self(),{double,5}}.
{<0.32.0>,{double,5}}
6> flush().
Shell got {<0.44.0>,10}
ok
7> Ser ! {self(),{double,7}}.
{<0.32.0>,{double,7}}
8> flush().
Shell got {<0.44.0>,14}
ok
9> Ser ! {self(),111}.
{<0.32.0>,111}
10> flush().
Shell got {<0.44.0>,error}
ok
```

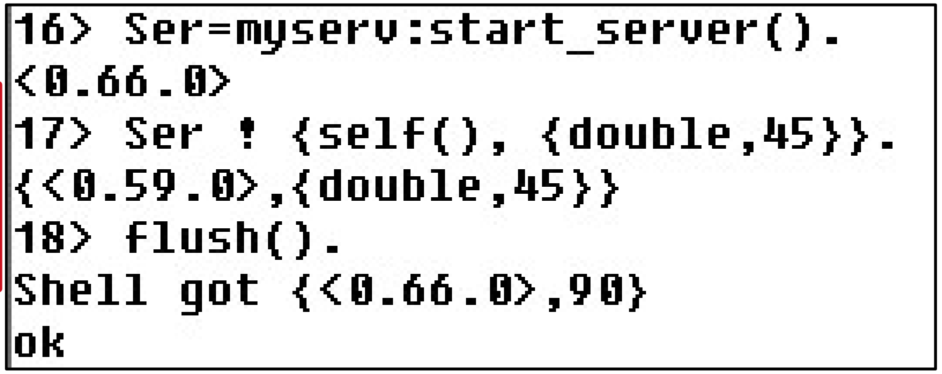
➢ Cilent-Server : functie pentru pornirea server-ului

```
-module(myserv).
-export([server_loop/0]).

server_loop() ->
  receive
       {From, {double, Number}} -> From ! {self(), Number*2},
                                                       server_loop() ;

        {From,_} -> From ! {self(),error},
                                 server_loop()
  end.
```



```
3> c(myserv).
{ok,myserv}
4> Ser=spawn(myserv, server_loop, []).
<0.44.0>
5> Ser ! {self(),{double,5}}.
{<0.32.0>,{double,5}}
6> flush().
Shell got {<0.44.0>,10}
ok
7> Ser ! {self(),{double,7}}.
{<0.32.0>,{double,7}}
8> flush().
Shell got {<0.44.0>,14}
ok
9> Ser ! {self(),111}.
{<0.32.0>,111}
10> flush().
Shell got {<0.44.0>,error}
ok
```

```
-export([start_server/0, server_loop/0]).
start_server() ->  spawn(myserv, server_loop, []).
```



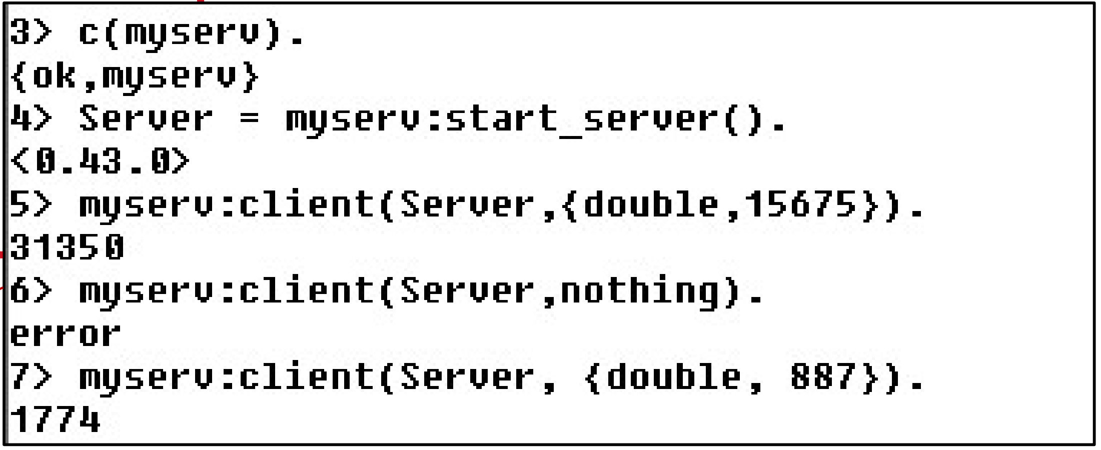
```
16> Ser=myserv:start_server().
<0.66.0>
17> Ser ! {self(), {double,45}}.
{<0.59.0>,{double,45}}
18> flush().
Shell got {<0.66.0>,90}
ok
```

➢ Cilent-Server: functia client

```
-module(myserv).
-export([start_server/0, server_loop/0,client/2]).


start_server() ->  spawn(myserv, server_loop, []).


server_loop() ->
  receive
     {From, {double, Number}} -> From ! {self(),(Number*2)},
                                                 server_loop() ;

     {From,_} -> From ! {self(),error},
                          server_loop()
  end.
```

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

functia **client intoarce raspunsul primit de la server**



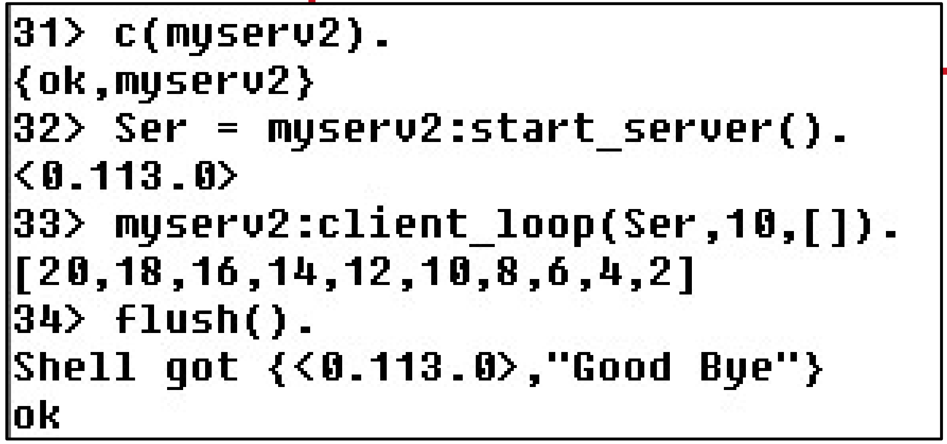
```
3> c(myserv).
{ok,myserv}
4> Server = myserv:start_server().
<0.43.0>
5> myserv:client(Server,{double,15675}).
31350
6> myserv:client(Server,nothing).
error
7> myserv:client(Server, {double, 887}).
1774
```

apelarea functiei client

➢ Client-Server

client_loop creaza mai multe procese client si intoarce lista rezultatelor

```
client_loop(Pid,0,L)  -> Pid! {self(),"Good Bye"},
                          L;

client_loop(Pid, X, L)  ->  R= client(Pid,{double,X}),
                            client_loop(Pid, X-1, L++[R]).
```


```
31> c(myserv2).
{ok,myserv2}
32> Ser = myserv2:start_server().
<0.113.0>
33> myserv2:client_loop(Ser,10,[]).
[20,18,16,14,12,10,8,6,4,2]
34> flush().
Shell got {<0.113.0>,"Good Bye"}
ok
```

- Client-Server

  client_loop creaza mai multi clienti si intoarce lista rezultatelor

```
client_loop(Pid,0,L)  -> Pid! {self(),"Good Bye"},
                          L;

client_loop(Pid, X, L)  ->  R= client(Pid,{double,X}),
                            client_loop(Pid, X-1, L++[R]).
```

Functiile client sunt executate **secvential!**

- Client-Server

  client_loop creaza mai multi clienti si intoarce lista rezultatelor

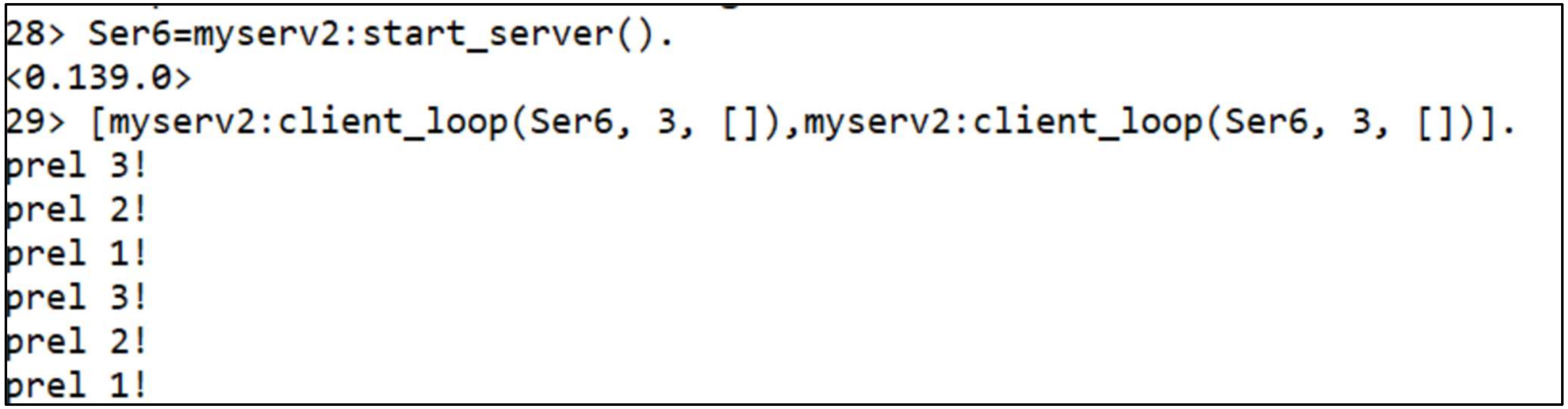
```
client_loop(Pid,0,L)  -> Pid! {self(),"Good Bye"},
                          L;

client_loop(Pid, X, L)  ->  R= client(Pid,{double,X}),
                            io:fwrite("prel ~w!~n", [N]),
                            client_loop(Pid, X-1, L++[R]).
```

Functiile client sunt executate **secvential!**

➢ Client-Server

client_loop creaza mai multi clienti si intoarce lista rezultatelor

```
client_loop(Pid,0,L)  -> Pid! {self(),"Good Bye"},
                         L;

client_loop(Pid, X, L)  ->  R= client(Pid,{double,X}),
                            io:fwrite("prel ~w!~n", [N]),

                            client_loop(Pid, X-1, L++[R]).
```



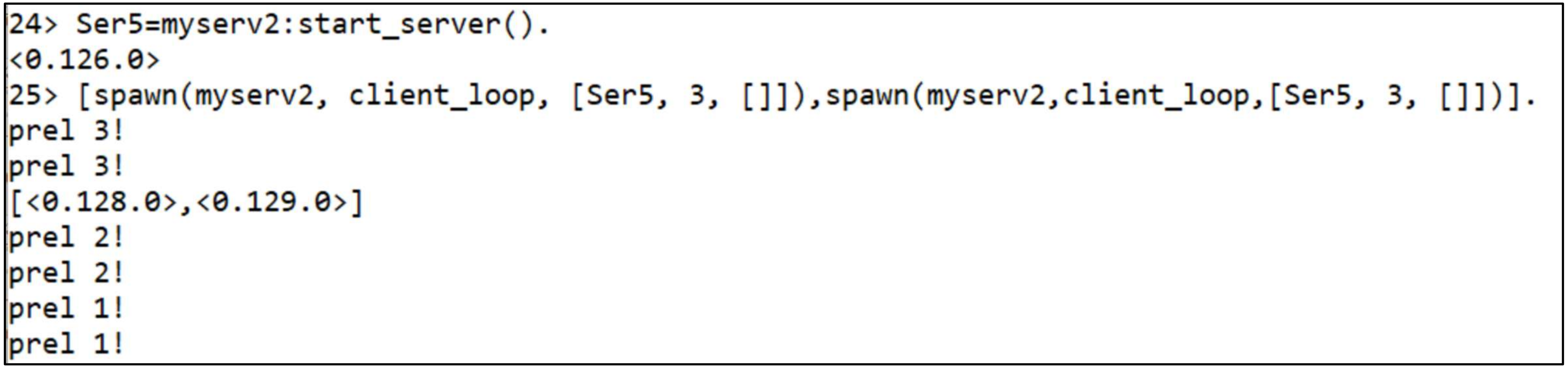
```
28> Ser6=myserv2:start_server().
<0.139.0>
29> [myserv2:client_loop(Ser6, 3, []),myserv2:client_loop(Ser6, 3, [])].
prel 3!
prel 2!
prel 1!
prel 3!
prel 2!
prel 1!
```

Functiile client sunt executate **secvential!**

➢ Client-Server

client_loop creaza mai multi clienti si intoarce lista rezultatelor

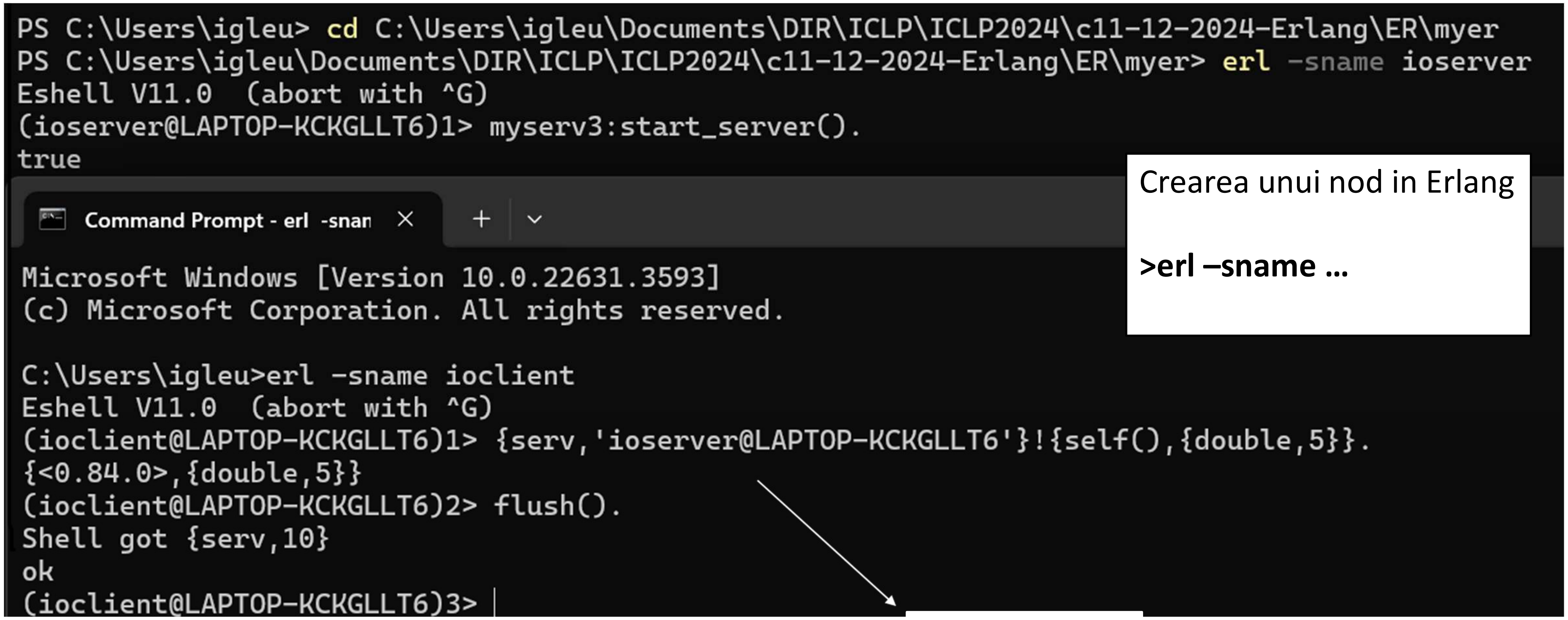
```
client_loop(Pid,0,L)  -> Pid! {self(),"Good Bye"},
                         L;

client_loop(Pid, X, L)  ->  R= client(Pid,{double,X}),
                            io:fwrite("prel ~w!~n", [N]),

                            client_loop(Pid, X-1, L++[R]).
```



```
24> Ser5=myserv2:start_server().
<0.126.0>
25> [spawn(myserv2, client_loop, [Ser5, 3, []]),spawn(myserv2,client_loop,[Ser5, 3, []])].
prel 3!
prel 3!
[<0.128.0>,<0.129.0>]
prel 2!
prel 2!
prel 1!
prel 1!
```

Functiile client sunt executate **in paralel!**

Distributed Erlang: programele ruleaza in noduri diferite



```
PS C:\Users\igleu> cd C:\Users\igleu\Documents\DIR\ICLP\ICLP2024\c11-12-2024-Erlang\ER\myer
PS C:\Users\igleu\Documents\DIR\ICLP\ICLP2024\c11-12-2024-Erlang\ER\myer> erl -sname ioserver
Eshell V11.0  (abort with ^G)
(ioserver@LAPTOP-KCKGLLT6)1> myserv3:start_server().
true
```

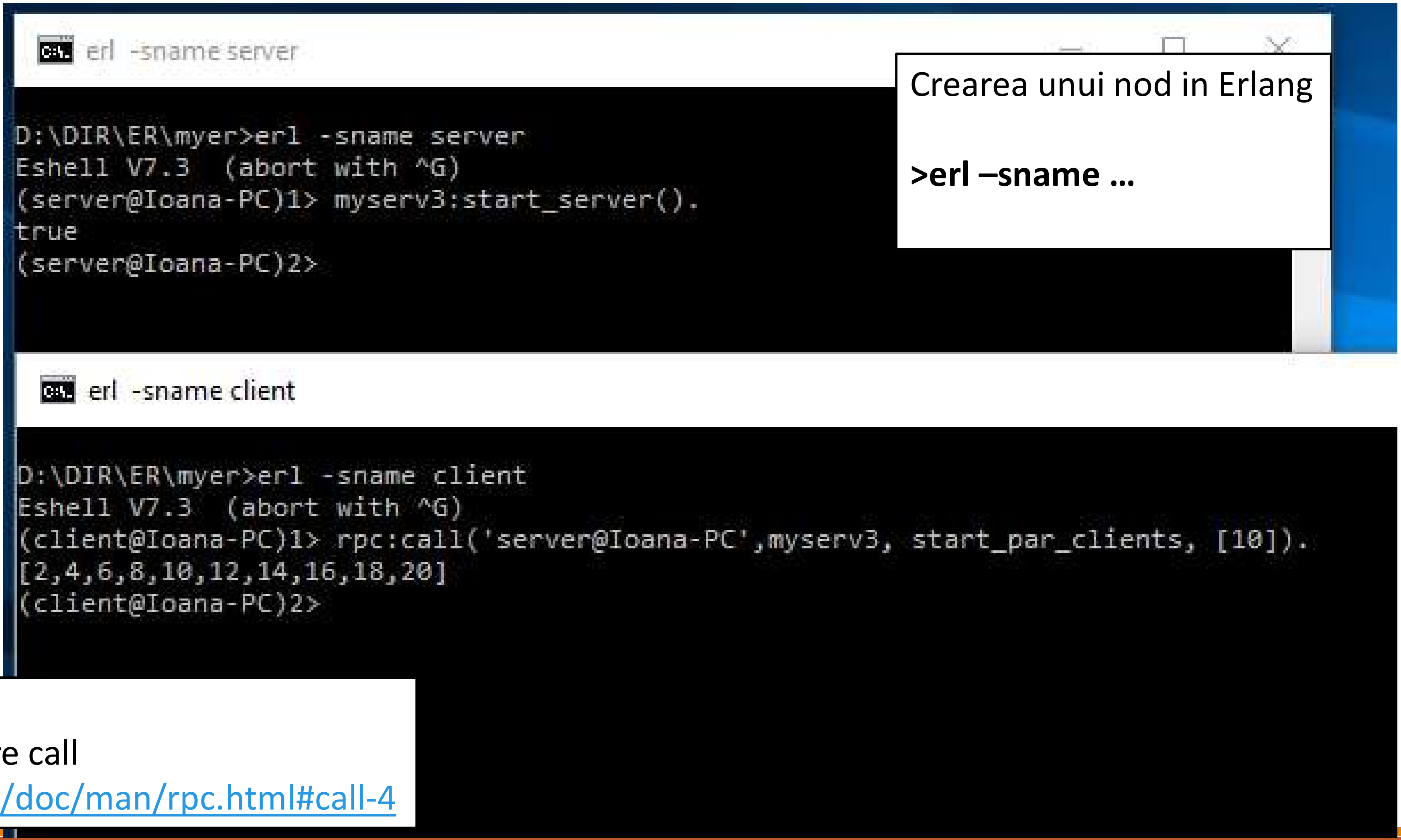
```
Microsoft Windows [Version 10.0.22631.3593]
(c) Microsoft Corporation. All rights reserved.

C:\Users\igleu>erl -sname ioclient
Eshell V11.0  (abort with ^G)
(ioclient@LAPTOP-KCKGLLT6)1> {serv,'ioserver@LAPTOP-KCKGLLT6'}!{self(),{double,5}}.
{<0.84.0>,{double,5}}
(ioclient@LAPTOP-KCKGLLT6)2> flush().
Shell got {serv,10}
ok
(ioclient@LAPTOP-KCKGLLT6)3>
```

Crearea unui nod in Erlang

>erl –sname …

Numele e atom!

Distributed Erlang: programele ruleaza in noduri diferite



```
D:\DIR\ER\myer>erl -sname server
Eshell V7.3  (abort with ^G)
(server@Ioana-PC)1> myserv3:start_server().
true
(server@Ioana-PC)2>
```

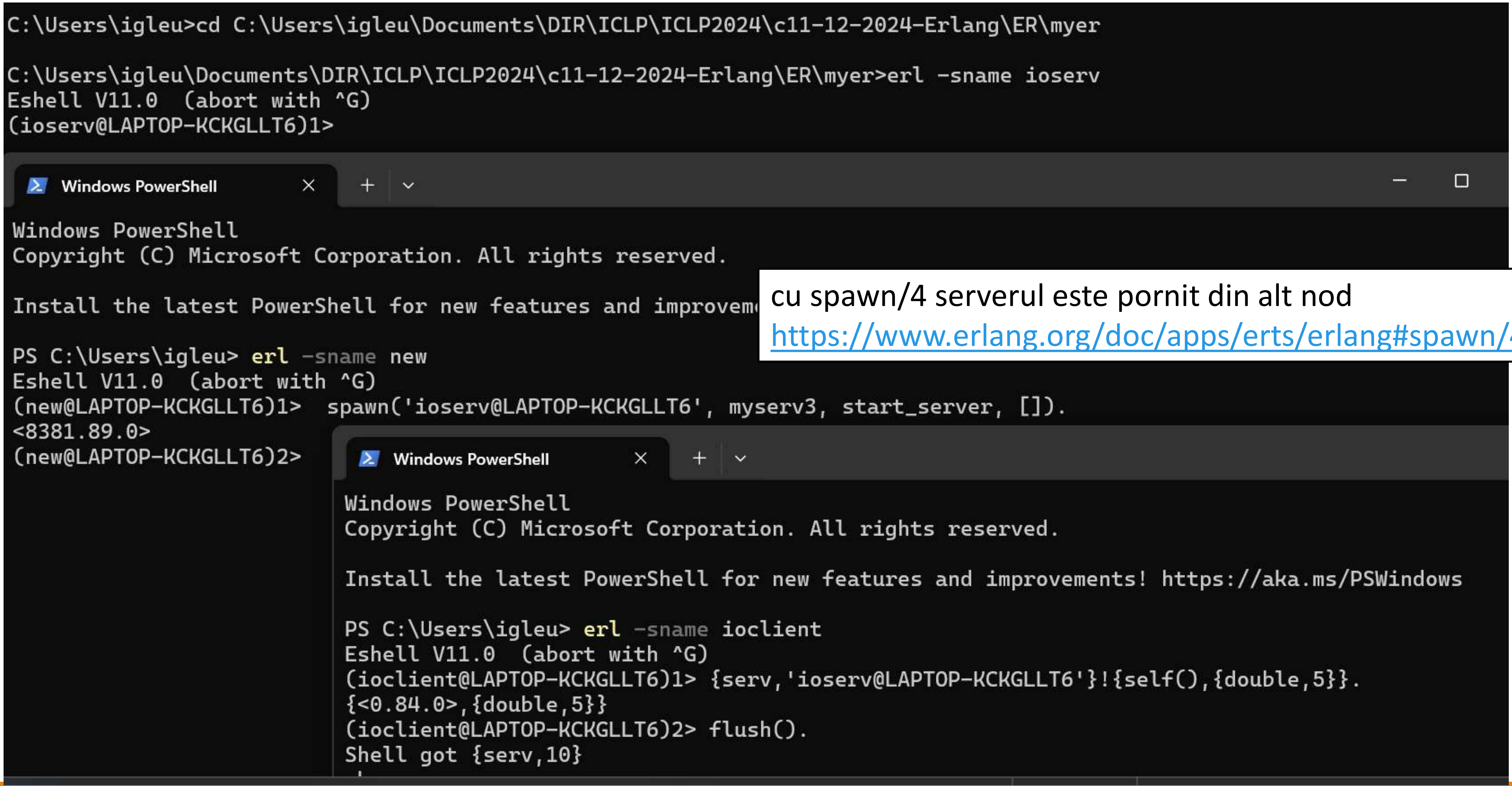
```
D:\DIR\ER\myer>erl -sname client
Eshell V7.3  (abort with ^G)
(client@Ioana-PC)1> rpc:call('server@Ioana-PC',myserv3, start_par_clients, [10]).
[2,4,6,8,10,12,14,16,18,20]
(client@Ioana-PC)2>
```

Crearea unui nod in Erlang

>erl –sname …

rpc:call
remote procedure call
http://erlang.org/doc/man/rpc.html#call-4

Distributed Erlang: programele ruleaza in noduri diferite



```
C:\Users\igleu>cd C:\Users\igleu\Documents\DIR\ICLP\ICLP2024\c11-12-2024-Erlang\ER\myer

C:\Users\igleu\Documents\DIR\ICLP\ICLP2024\c11-12-2024-Erlang\ER\myer>erl -sname ioserv
Eshell V11.0  (abort with ^G)
(ioserv@LAPTOP-KCKGLLT6)1>
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improveme

PS C:\Users\igleu> erl -sname new
Eshell V11.0  (abort with ^G)
(new@LAPTOP-KCKGLLT6)1>  spawn('ioserv@LAPTOP-KCKGLLT6', myserv3, start_server, []).
<8381.89.0>
(new@LAPTOP-KCKGLLT6)2>
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\igleu> erl -sname ioclient
Eshell V11.0  (abort with ^G)
(ioclient@LAPTOP-KCKGLLT6)1> {serv,'ioserv@LAPTOP-KCKGLLT6'}!{self(),{double,5}}.
{<0.84.0>,{double,5}}
(ioclient@LAPTOP-KCKGLLT6)2> flush().
Shell got {serv,10}
```

cu spawn/4 serverul este pornit din alt nod
https://www.erlang.org/doc/apps/erts/erlang#spawn/4