

# IV. rész

## Erlang alapok

- 1 Bevezetés
- 2 Cékla: deklaratív programozás C++-ban
- 3 Prolog alapok
- 4 Erlang alapok**
- 5 Haladó Prolog
- 6 Haladó Erlang

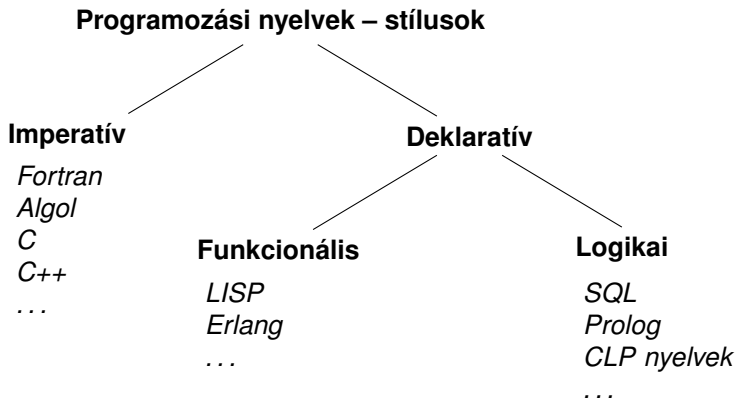
# Tartalom

4

## Erlang alapok

- **Bevezetés**
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

# Programozási nyelvek osztályozása



## Funkcionális programozás: mi az?

- Programozás *függvények alkalmazásával*
- Kevésbé elterjedten *applikatív programozásnak* is nevezik (vö. function application)
- A függvény: leképezés – az argumentumából állítja elő az eredményt  
A tiszta (matematikai) függvénynek nincs *mellékhatása*.

Példák funkcionális programozási nyelvekre, nyelvcsaládokra:

- Lisp, Scheme
- SML, Caml, Caml Light, OCaml, Alice
- Clean, Haskell
- Erlang
- F#

# Az Erlang nyelv

- 1985: megszületik „Ellemtelben” (Ericsson–Televerket labor)
  - Agner Krarup Erlang dán matematikus, ERicsson LANGUage
- 1991: első megvalósítás, első projektek
- 1997: első OTP (Open Telecom Platform)
- 1998-tól: nyílt forráskódú, szabadon használható
  
- Funkcionális alapú (Functionally based)
- Párhuzamos programozást segítő (Concurrency oriented)
- Gyakorlatban használt

„Programming is fun!”

# Szakirodalom angolul

- **Joe Armstrong: Programming Erlang. Software for a Concurrent World. The Pragmatic Bookshelf, 2007. ISBN-13 978-1-934356-00-5**  
<http://www.pragprog.com/titles/jaerlang/programming-erlang>
- Francesco Cesarini, Simon Thompson: Erlang Programming. O'Reilly, 2009. ISBN 978-0-596-51818-9 <http://oreilly.com/catalog/9780596518189/>
- Joe Armstrong, Robert Virding, Claes Wikström, Mike Williams: Concurrent Programming in Erlang. Second Edition. Prentice Hall, 1996. ISBN 0-13-508301-X. **Első része szabadon letölthető.**  
<http://erlang.org/download/erlang-book-part1.pdf>
- **On-line Erlang documentation** (Tutorial, Reference Manual stb.)  
<http://erlang.org/doc.html>  
Pl. listafüggvények: <http://www.erlang.org/doc/man/lists.html>
- Wikibooks on Erlang Programming  
[http://en.wikibooks.org/wiki/Erlang\\_Programming](http://en.wikibooks.org/wiki/Erlang_Programming)
- On-line help (csak unix/linux rendszeren) `man erl` vagy `erl -man <module>`

# Erlang emulátor

- Erlang shell (interaktív értelmező) indítása

```
$ erl
```

```
Erlang R13B03 (erts-5.7.4) [source] [smp:...
```

```
Eshell V5.7.4 (abort with ^G)
```

```
1>
```

```
1> 3.2 + 2.1 * 2.      % Lezárás és indítás „pont-bevitel”-lel!
```

```
7.4
```

```
2> atom.
```

```
atom
```

```
3> 'Atom'.
```

```
'Atom'
```

```
4> "string".
```

```
"string"
```

```
5> {ennes, 'A', a, 9.8}.
```

```
{ennes,'A',a,9.8}
```

```
6> [lista, 'A', a, 9.8].
```

```
[lista,'A',a,9.8]
```

```
7> q().
```

```
ok
```

# Erlang shell: parancsok

```
1> help().
** shell internal commands **
b()          -- display all variable bindings
e(N)         -- repeat the expression in query <N>
f()          -- forget all variable bindings
f(X)         -- forget the binding of variable X
h()          -- history
v(N)         -- use the value of query <N>
rr(File)     -- read record information from File (wildcards allowed)
...
** commands in module c **
c(File)      -- compile and load code in <File>
cd(Dir)      -- change working directory
help()       -- help info
l(Module)    -- load or reload module
lc([File])  -- compile a list of Erlang modules
ls()         -- list files in the current directory
ls(Dir)      -- list files in directory <Dir>
m()          -- which modules are loaded
m(Mod)       -- information about module <Mod>
pwd()        -- print working directory
q()          -- quit - shorthand for init:stop()
...
```



## Erlang shell: ^G és ^C

- ^G hatása

User switch command

```
--> h
c [nn]    - connect to job
i [nn]    - interrupt job
k [nn]    - kill job
j         - list all jobs
s         - start local shell
r [node]  - start remote shell
q         - quit erlang
? | h    - this message
--> c
```

- ^C hatása

```
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
(v)ersion (k)ill (D)b-tables (d)istribution
```

# Saját program lefordítása

## bevezeto.erl – Faktoriális számítása

```
-module(bevezeto).  % A modul neve (kötelező; modulnév = fájlnev)
-export([fac/1]).  % Látható függvények (praktikusan kötelező)

% @spec fac(N::integer()) -> F::integer().
% F = N! (F az N faktoriálisa).
fac(0) -> 1;      % ha az N=0 mintaillesztés sikeres
fac(N) -> N * fac(N-1).  % ha az N=0 mintaillesztés nem volt sikeres
```

- Fordítás, futtatás

```
1> c(bevezeto).
{ok,bevezeto}
2> bevezeto:fac(5).
120
3> fac(5).
** exception error: undefined shell command fac/1
4> bevezeto:fac(5)
4>
4> .
120
```

## Listakezelés – rövid példák (1)

```
1> L1 = [10,20,30].           % új változó kötése, '=' a mintaillesztés
[10,20,30]
2> H = hd(L1).              % hd: Built-in function (BIF)
10
3> T = tl(L1).              % tl: Built-in function
[20,30]
4> b().                      % kötött változók kiírása, lásd help().
H = 10
L1 = [10,20,30]
T = [20,30]
ok
5> T ::= [20|[30|[]]].      % egyenlőségvizsgálat
true
6> tl([]).
** exception error: bad argument
   in function tl/1
      called as tl([])
7> c(bevezeto).
{ok,bevezeto}
```

## Listakezelés – rövid példák (2)

bevezeto.erl – folytatás

*% sum(L) az L lista összege.*

sum([]) -> 0;

sum(L) -> H = hd(L), T = tl(L), H + sum(T).

*% append(L1, L2) az L1 lista L2 elé fűzve.*

append([], L2) -> L2;

append(L1, L2) -> [hd(L1)|append(tl(L1), L2)].

*% revapp(L1, L2) az L1 megfordítása L2 elé fűzve.*

revapp([], L2) -> L2;

revapp(L1, L2) -> revapp(tl(L1), [hd(L1)|L2]).

```
8> bevezeto:sum(L1).
```

```
60
```

```
9> bevezeto:append(L1, [a,b,c,d]).
```

```
[10,20,30,a,b,c,d]
```

```
10> bevezeto:revapp(L1, [a,b,c,d]).
```

```
[30,20,10,a,b,c,d]
```

## Prolog és Erlang: néhány eltérés és hasonlóság

- Néhány eltérés:

Prolog	Erlang
predikátum, kétféle érték	függvény, értéke tetsz. típusú
siker esetén változóbehelyettesítés	csak bemenő argumentum és visszatérési érték van
választási pontok, többféle megoldás	determinizmus, egyetlen mo.
összetett kifejezés (struktúra), a lista is	ennes, lista típusok (tuple, list)
operátor definiálása	-
egyesítés szimmetrikus	jobb oldalon tömör kif., bal oldalon mintakif.; őrfeltételekkel

- Néhány hasonlóság:

- a függvény is klózokból áll, kiválasztás mintaillesztéssel, sorrendben
- a függvényt is a funktora (pl. `bevezeto:fac/1`) azonosítja
- változóhoz csak egyszer köthető érték
- lista szintaxisa (de: Erlangban önálló típus), sztring (fűzér), atom

# Tartalom

4

## Erlang alapok

- Bevezetés
- **Típusok**
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

# Típusok

- Az Erlang erősen típusos nyelv, bár nincs típusdeklaráció
- A típusellenőrzés dinamikus és nem statikus
  - Alaptípusok

<i>magyarul</i>	<i>angolul</i>
Szám (egész, lebegőpontos)	Number (integer, float)
Atom	Atom
Függvény	Fun
Ennes (rekord)	Tuple (record)
Lista	List

- További típusok

Pid	Pid (Process identifier)
Port	Port
Hivatkozás	Reference
Bináris	Binary

# Atom

- Szövegkonstans (nem füzér!)
- Kisbetűvel kezdődő, bővített alfanumerikus<sup>1</sup> karaktersorozat, pl. `sicstus`, `erlang_OTP`
- Bármilyen<sup>2</sup> karaktersorozat is az, ha egyszeres idézőjelbe tesszük, pl. `'SICStus'`, `'erlang OTP'`, `'35 May'`
- Hossza tetszőleges, vezérlőkaraktereket is tartalmazhat, pl. `'ez egy hosszú atom, ékezetes betűkkel spékelve'`  
`'formázókarakterekkel \n\c\f\r'`
- Saját magát jelöli
- Hasonló a Prolog névkonstanshoz (atom)
- C++, Java nyelvekben a legközelebbi rokon: enum

---

<sup>1</sup>Bővített alfanumerikus: kis- vagy nagybetű, számjegy, aláhúzás (`_`), kukac (`@`).

<sup>2</sup>bármilyen latin-1 kódolású karaktert tartalmazó (R14B)



# Szám (number)

- Egész
  - Pl. 2008, -9, 0
  - Tetszőleges számrendszerben `radix#szám` alakban, pl. `2#101001`, `16#fe`
  - Az egész korlátlan pontosságú, pl.  
12345678901234567890123456789012345678901234
- Lebegőpontos
  - Pl. `3.14159`, `0.2e-22`
  - IEEE 754 64-bit
- Karakterkód
  - Ha nyomtatható: `$z`
  - Ha vezérlő: `$_n`
  - Oktális számmal: `$_012`

## Ennes (tuple)

- Rögzített számú, tetszőleges kifejezésből álló sorozat
- Példák: `{2008, erlang}`, `{'Joe', 'Armstrong', 16.99}`
- Nullás: `{}`
- Egyelemű ennes  $\neq$  ennes eleme, pl. `{elem}`  $\neq$  `elem`

## Lista (list)

- Korlátlan számú, tetszőleges kifejezésből álló sorozat
- Lineáris rekurzív adatszerkezet:
  - vagy üres (`[]` jellel jelöljük),
  - vagy egy elemből áll, amelyet egy lista követ: `[Elem|Lista]`
- Első elemét, ha van, a lista *fejének* nevezzük
- Első eleme utáni, esetleg üres részét a lista *farkának* nevezzük
  - Egyelemű lista: `[elem]`
  - Fejből-farokból létrehozott lista: `[elem|[]]`, `['első'|['második']]`
  - Többelemű lista:
    - `[elem,123,3.14,'elem']`
    - `[elem,123,3.14|['elem']]`
    - `[elem,123|[3.14,'elem']]`
- A konkatenáció műveleti jele: `++`  
`11> ['egy'|['két']] ++ [elem,123|[3.14,'elem']]`  
`[egy,két,elem,123,3.14,elem]`

## Füzér (string)

- Csak rövidítés, tkp. karakterkódok listája, pl.  
`"erl" ≡ [$e,$r,$l] ≡ [101,114,108]`
- Az Eshell a nyomtatható karakterkódok listáját füzérként írja ki:  
`12> [101,114,108]`  
`"erl"`
- Ha más érték is van a listában, listaként írja ki:  
`13> [31,101,114,108]`  
`[31,101,114,108]`  
`14> [a,101,114,108]`  
`[a,101,114,108]`
- Egymás mellé írással helyettesíthető, pl.  
`15> "erl" "ang".`  
`"erlang"`

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- **Az Erlang nyelv szintaxisának alapjai**
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

# Term, változó

## Term

- *Közelítő rekurzív definíció:* szám; atom; vagy termekből ennes- és listakonstruktorokkal felépített kifejezés.
- Néhány típussal (ref., port, pid, binary) egyelőre nem foglalkozunk
- Tovább nem egyszerűsíthető kifejezés, érték a programban
- Minden termnek van típusa
- Pl. 10 vagy {'Diák Detti', [{khf, [cekla, prolog, erlang, prolog]}}]  
Pl. nem term: 5 + 6
- Termek összehasonlítási sorrendje (v.ö. típusok)  
number < atom < ref. < fun < port < pid < tuple < list < binary

## Változó

- Nagybetűvel kezdődő, bővített alfanumerikus karaktersorozat, más szóval *név*
- A változó lehet *szabad* vagy *kötött*
- A szabad változónak nincs értéke, típusa
- A kötött változó értéke, típusa valamely konkrét term értéke, típusa
- Minden változóhoz *csak egyszer* köthető érték, azaz kötött változó nem kaphat értéket

# Kifejezés

- Lehet

- term
- változó
- minta

- Minta: term alakú kifejezés, amelyben szabad változó is lehet
- $\text{term} \subseteq \text{minta}^3$
- $\text{változó} \subseteq \text{minta}$

továbbá

- összetett kifejezés
  - szekvenciális kifejezés
  - függvényalkalmazás
  - egyéb: if, case, try/catch, catch stb.
- Kifejezés kiértékelése alapvetően: **mohó** (eager vagy strict evaluation).

```
16> Nevező = 0.
```

```
0
```

```
17> (Nevező > 0) and (1 / Nevező > 1).
```

```
** exception error: bad argument in an arithmetic expression
```

<sup>3</sup>néhány nem túl gyakorlatias ellenpéldától eltekintve

# Kifejezés: összetett és szekvenciális

## Összetett kifejezés

- Kiértékelhető műveleteket, függvényeket is tartalmazó, kifejezés, pl. `5+6`, vagy: `[{5+6, math:sqrt(2+fib(X))}, alma]`

## Szekvenciális kifejezés

- Kifejezések sorozata, szintaxisa:
  - `begin exp1, exp2, ..., expn end`
  - `exp1, exp2, ..., expn`
- A `begin` és `end` párt akkor kell kiírni, ha az adott helyen egyetlen kifejezésnek kell állnia
- Értéke az utolsó kifejezés értéke, `expn`
- ```
18> begin a, "a", 5, [1,2] end.  
[1,2]  
19> L2 = [10,20,30], H2 = hd(L2), T2 = tl(L2), H2 +  
bevezeto:sum(T2).  
60
```



# Kifejezés: függvényalkalmazás

## Függvényalkalmazás

- Szintaxisa

- `fnév(arg1, arg2, ..., argn)`  
vagy

- `modul:fnév(arg1, arg2, ..., argn)`

- Például

```
20> length([a,b,c]).
```

```
3
```

```
21> erlang:tuple_size({1,a,'A',"1aA"}).
```

```
4
```

## Függvénydeklaráció

- Egy vagy több, pontosvesszővel (;) elválasztott *klózból* állhat.
- Alakja:

```
fnév(A11, ..., A1m) [when ŐrSz1] -> SzekvenciálisKif1;  
...  
fnév(An1, ..., Anm) [when ŐrSzn] -> SzekvenciálisKifn.
```

- A függvényt a neve, az „aritása” (paramétereinek száma), valamint a moduljának a neve azonosítja.
- Az azonos nevű, de eltérő aritású függvények nem azonosak!
- Példák:

```
fac(0) -> 1;  
fac(N) -> N*fac(N-1).  
  
fac(0,R) -> R;  
fac(N,R) -> fac(N-1,N*R).
```

- (Őrök bemutatása kicsit később)

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- **Mintaillesztés**
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

## Minta, mintaillesztés (egyesítés)

- Minta (pattern): term alakú kifejezés, amelyben szabad változó is lehet
- Sikeres illesztés esetén a szabad változók kötötté válnak, értékük a megfelelő részkifejezés értéke lesz.

- A mintaillesztés (egyesítés) műveleti jele: =

Alakja: `MintaKif = TömörKif`

- Mintaillesztés  $\neq$  értékadás!

- Példák:

|                                       |                                                                                    |
|---------------------------------------|------------------------------------------------------------------------------------|
| <code>Pi = 3.14159</code>             | $\rightsquigarrow$ <sup>4</sup> <code>Pi</code> $\mapsto$ 3.14159 <sup>5</sup>     |
| <code>[H1 T1] = [1,2,3]</code>        | $\rightsquigarrow$ <code>H1</code> $\mapsto$ 1, <code>T1</code> $\mapsto$ [2,3]    |
| <code>[1,2 T2] = [1,2,3]</code>       | $\rightsquigarrow$ <code>T2</code> $\mapsto$ [3]                                   |
| <code>[H2 [3]] = [1,2,3]</code>       | $\rightsquigarrow$ <b>meghiúsulás, hiba</b>                                        |
| <code>{A1,B1} = {{a}, 'Beta'}</code>  | $\rightsquigarrow$ <code>A1</code> $\mapsto$ {a}, <code>B1</code> $\mapsto$ 'Beta' |
| <code>{{a},B2} = {{a}, 'Beta'}</code> | $\rightsquigarrow$ <code>B2</code> $\mapsto$ 'Beta'                                |

<sup>4</sup>`Kif`  $\rightsquigarrow$  jelentése: „`Kif` kiértékelése után”.

<sup>5</sup>`X`  $\mapsto$  `V` jelentése: „`X` a `V` értékhez van kötve”.

## Mintaillesztés függvény klózaira – 1. példa

- Függvény alkalmazásakor a klóz kiválasztása is mintaillesztéssel történik
- Máshol is, pl. a case vezérlési szerkezetnél is történik illesztés

### khf.erl – DP kisházik ellenőrzése

```
-module(khf).  
-compile(export_all).           % mindent exportál, csak teszteléshez!  
%-export([kiadott/1, ...]).    % tesztelés után erre kell cserélni
```

*% kiadott(Ny) az Ny nyelven kiadott kisházik száma*

```
kiadott(cekla)  -> 1;           % 1. klóz  
kiadott(prolog) -> 3;           % 2. klóz  
kiadott(erlang) -> 3.           % 3. klóz
```

```
2> khf:kiadott(cekla). % sikeres illesztés az 1. klózra
```

```
1
```

```
3> khf:kiadott(erlang). % sikertelen: 1-2. klóz, sikeres: 3. klóz
```

```
3
```

```
4> khf:kiadott(java). % 3 sikertelen illesztés után hiba
```

```
** exception error: no function clause matching ...
```

## Mintaillesztés függvény klózaira – 2. példa

- Hányszor szerepel egy elem egy listában? Első megoldásunk:

khf.erl – folytatás

```
% @spec elofordul1(E::term(), L::[term()]) -> N::integer().
% E elem az L listában N-szer fordul elő.
elofordul1(E, [])          -> 0;                               % 1.
elofordul1(E, [E|Farok])  -> 1 + elofordul1(E, Farok);      % 2.
elofordul1(E, [_Fej|Farok]) -> elofordul1(E, Farok).        % 3.
```

```
5> khf:elofordul1(a, [a,b,a,1]). % 2. klóz, majd 3., 2., 3., 1.
2
6> khf:elofordul1(java, [cekla,prolog,prolog]). % 3., 3., 3., 1.
0
```

- A minták összekapcsolhatóak, az E változó több argumentumban is szerepel: `elofordul1(E, [E|Farok]) -> ...`
- Számít a klózok sorrendje, itt pl. a 3. általánosabb, mint a 2.!

## Mintaillesztés függvény klózáira – 3. példa

- Teljesítette-e egy hallgató a khf követelményeket?

```
7> Hallgato1 = {'Diák Detti',  
               [{khf, [cekla,prolog,erlang,prolog]},  
               {nzh, 59}]}.
```

### khf.erl – folytatás

```
% @spec megfelelt(K::kovetelmeny(), H::hallgato()) -> true | false.  
megfelelt(khf, {_Nev, [{khf, L}|_]}) ->  
    C = elofordul1(cekla, L),  
    P = elofordul1(prolog, L),  
    E = elofordul1(erlang, L),  
    (P >= 1) and (E >= 1) and (C + P + E >= 3);  
megfelelt(K, {_Nev, [_|F]}) ->  
    megfelelt(K, {_Nev, F});  
megfelelt(_, {_, []}) ->  
    false.
```

## „Biztonságos” illesztés: ha egyik mindig sikerül

- Mit kezdünk a kiadott(java) kiértékelésekor keletkező hibával?
- Erlangban gyakori: jelezzük a sikert vagy a hibát az eredményben

khf.erl – folytatás

```
% @spec safe_kiadott(Ny::atom()) -> {ok, Db::integer()} | error.  
% Az Ny nyelven Db darab kisházit adtak ki.  
safe_kiadott(cekla) -> {ok, 1};  
safe_kiadott(prolog) -> {ok, 3};  
safe_kiadott(erlang) -> {ok, 3};  
safe_kiadott(_Ny) -> error.    % e klóz mindig illeszthető
```

- Az ok és az error atomokat konvenció szerint választottuk
- Kötés: ha a minta egyetlen szabad változó (\_Ny), az illesztés sikeres
- De hogy férjük hozzá az eredményhez?

```
8> khf:safe_kiadott(cekla).  
{ok, 1}  
9> khf:safe_kiadott(java).  
error
```



## Feltételes kifejezés mintaillesztéssel (case)

- `case Kif of`

```
Minta1 [when ŐrSz1] -> SzekvenciálisKif1;
```

```
...
```

```
Mintan [when ŐrSz1n] -> SzekvenciálisKifn
```

```
end.
```

- Kiértékelés: balról jobbra

- Értéke: az első illeszkedő minta utáni szekvenciális kifejezés

- Ha nincs ilyen minta, hibát jelez

```
10> X=2, case X of 1 -> "1"; 3 -> "3" end.
```

```
** exception error: no case clause matching 2
```

```
11> X=2, case X of 1 -> "1"; 2 -> "2" end.
```

```
"2"
```

```
12> Y=fagylalt, 3 * case Y of fagylalt -> 100; tolcser -> 15 end.
```

```
300
```

```
13> Z=kisauto, case Z of fagylalt -> 100;
```

```
13> tolcser -> 15;
```

```
13> Barmi -> 99999 end.
```

```
99999
```

## case példa

- Az adott nyelvből hány százalékot adtuk be?

khf.erl – folytatás

```
% @spec safe_teljesitmeny(Nyelv::atom(), Beadott_db::integer()) ->
%   {ok, Teljesitmeny::float()} | error.
safe_teljesitmeny(Nyelv, Beadott_db) ->
  case safe_kiadott(Nyelv) of
    {ok, Kiadott_db} -> {ok, Beadott_db / Kiadott_db};
    error             -> error
  end.
```

- Függvény klózai összevonhatóak a case segítségével:

|                       |         |                |
|-----------------------|---------|----------------|
| kiadott(cekla) -> 1;  |         | kiadott(Ny) -> |
| kiadott(prolog) -> 3; |         | case Ny of     |
| kiadott(erlang) -> 3. | helyett | cekla -> 1;    |
|                       | írható: | prolog -> 3;   |
|                       |         | erlang -> 3    |
|                       |         | end.           |

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- **Magasabbrendű függvények, függvényérték**
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

# Függvényérték

- A funkcionális nyelvekben a függvény is *érték*:
  - leírható (jelölhető)
  - van típusa
  - névhez (változóhoz) köthető
  - adatszerkezet eleme lehet
  - **paraméterként átadható**
  - **függvényalkalmazás eredménye lehet** (zárójelezni kell!)

- Névtelen függvény (függvényjelölés) mint érték

```
fun (A11, ..., A1m) [when ŐrSz1] -> SzekvenciálisKif1;
    ...;
    (An1, ..., Anm) [when ŐrSzn] -> SzekvenciálisKifn
end.
```

- Már deklarált függvény mint érték

```
fun Modul:Fnev/Aritas % például fun bevezeto:sum/1
fun Fnev/Aritas % ha az Fnev „látható”, pl. modulból
```

„Programming is fun!”

## Függvényérték: példák

```
2> Area1 = fun ({circle,R})    -> R*R*3.14159;
           ({rectan,A,B})     -> A*B;
           ({square,A})      -> A*A
           end.
#Fun<erl_eval.6.13229925>
3> Area1({circle,2}).
12.56636
4> Osszeg = fun bevezeto:sum/1.
#Fun<bevezeto.sum.1>
5> Osszeg([1,2]).
3
6> fun bevezeto:sum/1([1,2]).
3
7> F1 = [Area1, Osszeg, fun bevezeto:sum/1, 12, area].
[#Fun<erl_eval.6.13229925>,#Fun<bevezeto.sum.1>,...]
8> (hd(F1))({circle, 2}). % külön zárójelezni kell!
12.56636
% hd/1 itt magasabbrendű függvény, zárójelezni kell értékét
```

## Magasabb rendű függvények alkalmazása – map, filter

- **Magasabb rendű függvény:** paramétere vagy eredménye függvény
- **Leképzés:** `lists:map(Fun, List)` A List lista Fun-nal transzformált elemeiből álló lista

```
9> lists:map(fun erlang:length/1, ["alma", "korte"]).
[4,5]           % erlang:length/1: Built-In Function, lista hossza
10> lists:map(Osszeg, [[10,20], [10,20,30]]).
[30,60]
11> L = [{'Diák Detti', [{khf, [...]}]}, {'Lusta Ludvig', []}].
[{'Diák Detti', [{khf, [...]}]}, {'Lusta Ludvig', []}]
12> lists:map(fun(Hallg) -> khf:megfelelt(khf, Hallg) end, L).
[true,false]
```

- **Szűrés:** `lists:filter(Pred, List)`  
A List lista Pred-et kielégítő elemeinek listája

```
13> lists:filter(fun erlang:is_number/1, [x, 10, L, 20, {}]).
[10,20]
14> lists:filter(fun(Hallg) -> khf:megfelelt(khf, Hallg) end, L).
[{'Diák Detti', [{khf, [...]}]}
```

## Magasabb rendű függvények alkalmazása – filter példa

- Hányszor szerepel egy elem egy listában? Új megoldásunk:

khf.erl – folytatás

```
% @spec elofordul2(E::term(), L::[term()]) -> N::integer().
% E elem az L listában N-szer fordul elő.
```

```
elofordul2(Elem, L) ->
  length(filter(fun(X) -> X == Elem end, L)).
```

```
15> khf:elofordul2(prolog, [cekla,prolog,prolog]).
```

```
2
```

- A névtelen függvényben felhasználhatjuk az Elem lekötött változót!
- A filter/2 egy lehetséges megvalósítása:

```
filter(_, []) -> [];
filter(P, [Fej|Farok]) -> case P(Fej) of
                            true -> [Fej|filter(P, Farok)];
                            false -> filter(P, Farok)
                        end.
```

- Fejtörő: hogyan lehetne megvalósítani a filtert case nélkül, klózillesztéssel?

## Redukálás a fold függvényekkel

- Jobbról balra haladva: `lists:foldr(Fun, Acc, List)`
- Balról jobbra haladva: `lists:foldl(Fun, Acc, List)`
- A `List` lista elemeiből és az `Acc` elemből a kétoperandusú `Fun`-nal képzett érték

```
lists:foldr(fun(X, Acc) -> X - Acc end, 0, [1,2,3,4]) ≡ -2
```

```
lists:foldl(fun(X, Acc) -> X - Acc end, 0, [1,2,3,4]) ≡ 2
```

- Példa `foldr` kiértékelési sorrendjére:  $1 - (2 - (3 - (4 - \text{Acc}))) = -2$

Példa `foldl` kiértékelési sorrendjére:  $4 - (3 - (2 - (1 - \text{Acc}))) = 2$

---

```
% plus(X, Sum) -> X + Sum.
```

---

|   |                                                                                    |  |                                                                                                        |
|---|------------------------------------------------------------------------------------|--|--------------------------------------------------------------------------------------------------------|
| R | <pre>sum(Acc, []) -&gt;   Acc; sum(Acc, [H T]) -&gt;   plus(H, sum(Acc, T)).</pre> |  | <pre>foldr(Fun, Acc, []) -&gt;   Acc; foldr(Fun, Acc, [H T]) -&gt;   Fun(H, foldr(Fun, Acc, T)).</pre> |
|---|------------------------------------------------------------------------------------|--|--------------------------------------------------------------------------------------------------------|

---

|   |                                                                                    |  |                                                                                                         |
|---|------------------------------------------------------------------------------------|--|---------------------------------------------------------------------------------------------------------|
| L | <pre>sum(Acc, []) -&gt;   Acc; sum(Acc, [H T]) -&gt;   sum(plus(H, Acc), T).</pre> |  | <pre>foldl(Fun, Acc, []) -&gt;   Acc; foldl(Fun, Acc, [H T]) -&gt;   foldl(Fun, Fun(H, Acc), T)).</pre> |
|---|------------------------------------------------------------------------------------|--|---------------------------------------------------------------------------------------------------------|



# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- **Listanézet**
- Műveletek, beépített függvények
- Őr
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

## Listanézet (List Comprehensions)

- Listanézet (List Comprehensions): `[Kif || Minta <- Lista, Feltétel]`  
*Közelítő definíció:* **Kif** kifejezések *listája*, ahol a **Minta** a **Lista** olyan eleme, melyre **Feltétel** igaz.
- Feltétel** tetszőleges logikai kifejezés lehet. A **Mintában** előforduló változónevek elfedik a listakifejezésen kívüli azonos nevű változókat.
- Kis példák
 

```
1> [2*X || X <- [1,2,3]].      % { 2·x | x ∈ {1,2,3} }
[2,4,6]
2> [2*X || X <- [1,2,3], X rem 2 /= 0, X > 2].
[6]
3> lists:seq(1,3).           % egészek 1-től 3-ig
[1,2,3]
4> [{X,Y} || X <- [1,2,3,4], Y <- lists:seq(1,X)].
[{1,1},
 {2,1},{2,2},
 {3,1},{3,2},{3,3},
 {4,1},{4,2},{4,3},{4,4}]
```
- Pontos szintaxis: `[X || Q1, Q2, ...]`, ahol **X** tetszőleges kifejezés, **Q<sub>i</sub>** lehet generátor (`Minta <- Lista`) vagy szűrőfeltétel (predikátum)

## Listanézet: példák

- Pitagoraszai számhármások, melyek összege legfeljebb  $N$

```
pitag(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
  ].
```

- Hányszor fordul elő egy elem egy listában?

```
elofordul3(Elem, L) ->
  length([X || X <- L, X==Elem]).
```

- A khf követelményeket teljesítő hallgatók

```
L = [{'Diák Detti', [{khf, [...]}]}, {'Lusta Ludvig', []}],
[Nev || {Nev, M} <- L, khf:megfelelt(khf, {Nev, M})].
```

## Listanézet: érdekes példák

- Quicksort

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

- Permutáció

```
perms([]) -> [[]];  
perms(L) ->  
  [[H|T] || H <- L, T <- perms(L--[H])].
```

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- **Műveletek, beépített függvények**
- Ör
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

## Listaműveletek

- Listák összefűzése ( $A_s \oplus B_s$ ):  $A_s++B_s$  vagy `lists:append(A_s,B_s)`  
 $C_s = A_s++B_s \rightsquigarrow C_s \mapsto$  az  $A_s$  összes eleme a  $B_s$  elé fűzve az eredeti sorrendben
- Példa  

```
1> [a,'A',[65]]++[1+2,2/1,'A'] .
[a,'A',"A",3,2.0,'A']
```
- Listák különbsége:  $A_s--B_s$  vagy `lists:subtract(A_s,B_s)`  
 $C_s = A_s--B_s \rightsquigarrow C_s \mapsto$  az  $A_s$  olyan másolata, amelyből ki van hagyva a  $B_s$ -ben előforduló összes elem balról számított első előfordulása, feltéve, hogy volt ilyen elem  $A_s$ -ben
- Példa  

```
1> [a,'A',[65],'A']--["A",2/1,'A'] .
[a,'A']
2> [a,'A',[65],'A']--["A",2/1,'A',a,a,a] .
['A']
3> [1,2,3]--[1.0,2]. % erős típusosság: 1 ≠ 1.0
[1,3]
```

# Aritmetikai műveletek

- Matematikai műveletek

- Előjel: +, - (precedencia: 1)
- Multiplikatív: \*, /, div, rem (precedencia: 2)
- Additív: +, - (precedencia: 3)


- Bitműveletek

- bnot, band (precedencia: 2)
- bor, bxor, bsl, bsr (precedencia: 3)

- Megjegyzések

- +, -, \* és / egész és lebegőpontos operandusokra is alkalmazhatók
- +, - és \* eredménye egész, ha mindkét operandusuk egész, egyébként lebegőpontos
- / eredménye mindig lebegőpontos
- div és rem, valamint a bitműveletek operandusai csak egészek lehetnek

# Relációk

- Termék összehasonlítási sorrendje (v.ö. típusok):  
number < atom < ref. < fun < port < pid < tuple < list < binary
- Kisebb-nagyobb reláció  
<, =<, >=, >
- Egyenlőségi reláció (aritmetikai egyenlőségre is):  
==, /= **ajánlás: helyette azonosan egyenlőt használjunk!**
- Azonosan egyenlő (különbséget tesz integer és float közt):  
:=, /= Péllda: 1> 5.0 := 5.  
false
- Az összehasonlítás eredménye a true vagy a false atom
-  Lebegőpontos értékre kerülendő:  
1> (10.1 - 9.9) \* 10 == 2.  
false  
2> 0.000000000000000001 + 1 == 1.  
Elrettentő példák: true
- Kerekítés (float  $\mapsto$  integer), explicit típuskonverzió (integer  $\mapsto$  float):  
erlang:trunc/1, erlang:round/1, erlang:float/1



# Logikai műveletek

- Logikai művelet:  
not, and, or, xor
- Csak a true és false atomokra alkalmazhatóak
- Lusta kiértékelésű („short-circuit”) logikai művelet:  
andalso, orelse
- Példák:  
1> `false and (3 div 0 == 2).`  
`** exception error: bad argument in an arithmetic expression`  
  
2> `false andalso (3 div 0 == 2).`  
`false`

## Beépített függvények (BIF)

- BIF (Built-in functions)
  - a futtatórendszerbe beépített, rendszerint C-ben írt függvények
  - többségük az **erts**-könyvtár erlang moduljának része
  - többnyire rövid néven (az erlang: modulnév nélkül) hívhatók
- Az alaptípusokon alkalmazható leggyakoribb BIF-ek:
  - Számok:  
`abs(Num)`, `trunc(Num)`, `round(Num)`, `float(Num)`
  - Lista:  
`length(List)`, `hd(List)`, `tl(List)`
  - Ennes:  
`tuple_size(Tuple)`,  
`element(Index, Tuple)`,  
`setelement(Index, Tuple, Value)`  
Megjegyzés:  $1 \leq \text{Index} \leq \text{tuple\_size}(\text{Tuple})$

## További BIF-ek

- Rendszer:  
`date()`, `time()`, `erlang:localtime()`, `halt()`
- Típusvizsgálat
  - `is_integer(Term)`, `is_float(Term)`,
  - `is_number(Term)`, `is_atom(Term)`,
  - `is_boolean(Term)`,
  - `is_tuple(Term)`, `is_list(Term)`,
  - `is_function(Term)`, `is_function(Term, Arity)`
- Típuskonverzió
  - `atom_to_list(Atom)`, `list_to_atom(String)`,
  - `integer_to_list(Int)`, `list_to_integer(String)`,  
`erlang:list_to_integer(String, Base)`,
  - `float_to_list(Float)`, `list_to_float(String)`,
  - `tuple_to_list(Tuple)`, `list_to_tuple(List)`
- Érdekeség: a BIF-ek mellett megtalálhatóak az operátorok az `erlang` modulban, lásd az `m(erlang). kimenetét`, pl. `fun erlang:'*'/2(3,4)`.

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- **Őr**
- Típusspecifikáció
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

## Őrszekvencia (Guard sequence)

- Nézzük újra a következő definíciót:

```
fac(0) -> 1;  
fac(N) -> N * fac(N-1).
```

- Mi történik, ha megváltoztatjuk a klózik sorrendjét?
- Mi történik, ha -1-re alkalmazzuk?
- És ha 2.5-re?

A baj az, hogy a `fac(N) -> ...` klóz túl általános.

- Megoldás: korlátozzuk a mintaillesztést őrszekvencia alkalmazásával

```
fac(0) ->  
    1;  
fac(N) when is_integer(N), N>0 ->  
    N*fac(N-1).
```

# Ismétlés: függvénydeklaráció, case

- Függvénydeklaráció:

```
fnév(A11, ..., A1m) [when ŐrSz1] -> SzekvenciálisKif1;  
...  
fnév(An1, ..., Anm) [when ŐrSzn] -> SzekvenciálisKifn.
```

- Feltételes mintaillesztés (case):

```
case Kif of  
  Minta1 [when ŐrSz1] -> SzekvenciálisKif1;  
  ...  
  Mintan [when ŐrSz1n] -> SzekvenciálisKifn  
end.
```

## Őr, őrszekvencia, őrkifejezés

- Az őrszekvenciával olyan tulajdonságot írunk elő, amit strukturális mintaillesztéssel nem tudunk leírni
- Az őrszekvenciát a `when` kulcsszó vezeti be
- Az őrszekvenciában előforduló összes változónak *kötöttnek* kell lennie

Elnevezések:

- **Őrkifejezés (Guard expression):** korlátozott Erlang-kifejezés, mellékhatás nélküli
- **Őr (Guard):** egyetlen őrkifejezés vagy őrkifejezések vesszővel (`,`) elválasztott sorozata
  - `true`, ha az összes őrkifejezés `true` (ÉS-kapcsolat)
- **Őrszekvencia (Guard sequence):** egyetlen őr vagy örök pontosvesszővel (`;`) elválasztott sorozata
  - `true`, ha legalább egy őr `true` (VAGY-kapcsolat)
  - Ha értéke `true`  $\rightsquigarrow$  *sikerül*, bármely más term  $\rightsquigarrow$  *meghiúsul*
  - Sokszor helytelenül őrnek rövidítik; mentség: az őr tipikusan elegendő, őrszekvencia ritka

# Örkifejezés, mintaillesztés

Örkifejezés:

- Örkifejezés  $\subset$  Erlang-kifejezés
- Garantáltan mellékhatás nélküli, hatékonyan kiértékelhető
- Vagy sikerül, vagy meghiúsul
- Hibát (kivételt) **nem** jelezhet; ha hibás az argumentuma, meghiúsul

A mintaillesztés lépései klózválasztásnál, case-nél:

- Strukturális mintaillesztés (hasonló a Prolog illesztésére)
- Órszekvencia kiértékelése



# Örkifejezés

Örkifejezés lehet:

- Term (vagyis konstans érték)
- Kötött változó
- Örkifejezésekből aritmetikai, összehasonlító és logikai műveletekkel felépített kifejezés
- Bizonyos BIF-ek örkifejezéssel paraméterezve:
  - Típust vizsgáló predikátumok (`is_TÍPUS`)
  - `abs(Number)` `round(Number)` `trunc(Number)` `float(Term)`  
`element(N, Tuple)` `tuple_size(Tuple)`  
`hd(List)` `length(List)` `tl(List)`  
`bit_size(Bitstring)` `byte_size(Bitstring)` `size(Tuple|Bitstring)`  
`node()` `node(Pid|Ref|Port)` `self()`

Örkifejezés **nem** lehet:

- Függvényalkalmazás, mert esetleg mellékhatása lehet vagy lassú
- `++ (lists:append/2)`, `-- (lists:subtract/2)`

## Örszekvencia: példák

`orok.erl` – *kategoria(V) a V term egy lehetséges osztályozása.*

```
kategoria(V) ->
  case V of
    X when is_atom(X) ->
      atom;
    X when is_number(X), X < 0 ->
      negativ_szam;
    X when is_integer(X) ;
      is_float(X), abs(X-round(X)) < 0.0001 ->
      kerek_szam;
    X when is_list(X), length(X) > 5 ->
      hosszu_lista;
    ...
```

```
2> orok:kategoria(true).
```

```
atom
```

```
3> [{K,orok:kategoria(K)} || K <- [haho, -5, 5.000001, "kokusz"] ].
```

```
[{haho,atom}, {-5,negativ_szam}, {5.000001,kerek_szam},
 {"kokusz",hosszu_lista}]
```

## Órszekvencia: példák – folytatás

orok.erl – kategoria(V) folytatása

```

...
{X,Y,Z} when X*X+Y*Y == Z*Z, is_integer(Z) ;
           Z*Z+Y*Y == X*X, is_integer(X) ;
           X*X+Z*Z == Y*Y, is_integer(Y) ->
    pitagoraszi_szamharmas;
{Nev, []} when is_atom(Nev) ->
    talan_hallgato;
{Nev, [{Tipus,_}|_]} when is_atom(Nev), is_atom(Tipus) ->
    talan_hallgato;
[Ny1|_] when Ny1==cekla ; Ny1==prolog ; Ny1==erlang ->
    talan_programozasi_nyelvek_listaja;
{tort, Sz, N} when abs(Sz div N) >= 0 -> % Ha Sz vagy N nem
    racionalis; % egész, vagy ha N:=0, hiba miatt megghiúsul
_ -> egyeb
end.

```

```

4> [orok:kategoria(K) || K <- [{3,5,4}, {'D.D.',[],}, {tort,1,a}]].
[pitagoraszi_szamharmas,talan_hallgato,egyeb]

```

## Feltételes kifejezés őrsekvenciával

- `if`

```
ÖrSz1 -> SzekvenciálisKif1;
```

```
...
```

```
ÖrSzn -> SzekvenciálisKifn
```

```
end.
```

- Kiértékelés: balról jobbra.
- Értéke: az első teljesülő őrsekvencia utáni szekvenciális kifejezés
- Ha nincs ilyen őrsekvencia, futáskor hibát jelez.
- Példák

```
1> X=2.
2> if X<2 -> "<"; X>2 -> ">" end.
** exception error: no true branch...
3> if X<2 -> "<"; X>=2 -> ">=" end.
">="
4> if X<2 -> "<"; true -> ">=" end.
">="
```

### khf.erl – folytatás

```
elofordul4(_, []) -> 0;
elofordul4(E, [Fej|Farok]) ->
    if
        Fej == E -> 1;
        true      -> 0
    end
+ elofordul4(Elem, Farok).
```

## Az if a case speciális esete

- case: kifejezést illeszt mintákra őr szekvenciával, if: csak őr szekvenciák
- if helyettesítése case-zel (az Alapértelmezés sora opcionális):

```

case 1 of % _=1 mindig sikeres lenne
  _ when ŐrSz1 -> Kif1;
  ...
  _ when ŐrSzn -> Kifn;
  _ -> Alapértelmezés
end

                                     ≡

if
  ŐrSz1 -> Kif1;
  ...
  ŐrSzn -> Kifn;
  true -> Alapért
end

```

- Fordítva: pl. használhatunk-e case helyett if-et?

```

filter(_, []) -> [];
filter(P, [Fej|Farok]) -> case P(Fej) of
  true -> [Fej|filter(P,Farok)];
  false -> filter(P,Farok)
end.

```

Vigyázat! if P(Fej) -> Kif... hibás lenne, űrben nem lehet függvény „illegal guard expression”

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- **Típus-specifikáció**
- Kivételkezelés
- Rekord
- Gyakori könyvtári függvények

# Típus-specifikáció

- Csak *dokumentációs konvenció*, nem nyelvi elem az Erlangban
- Készültek programok a típus-specifikáció és a programkód összevetésére
- A *typeName* típust is jelöljük: `typeName()`.
- Típusok: előre definiált és felhasználó által definiált

## Előre definiált típusok

- `any()`, `term()`: bármely Erlang-típus
- `atom()`, `binary()`, `float()`, `function()`, `integer()`, `pid()`, `port()`, `reference()`: Erlang-alaptípusok
- `bool()`: a `false` és a `true` atomok
- `char()`: az `integer` típus karaktereket ábrázoló része
- `iolist() = [char() | binary() | iolist()]`<sup>6</sup>: karakter-io
- `tuple()`: ennestípus
- `list(L)`: `[L]` listatípus szinonimája
- `nil()`: `[]` üreslista-típus szinonimája
- `string()`: `list(char())` szinonimája
- `deep_string() = [char() | deep_string()]`
- `none()`: a „nincs típusa” típus; nem befejeződő függvény „eredményének” megjelölésére

---

<sup>6</sup> ... | ... választási lehetőség a szintaktikai leírásokban.



## Új (felhasználó által definiált) típusok

- Szintaxis: `@type newType() = Típuskifejezés.`
- Típuskifejezés az előre definiált típus, a felhasználó által definiált típus és a típusváltozó
- Uniótípus  
`T1 | T2` típuskifejezés, ha `T1` és `T2` típuskifejezések  
`% @type nyelv() = cekla | prolog | erlang.`
- Listatípus  
`[T]` típuskifejezés, ha `T` típuskifejezés  
`% @type nyelvlista() = [nyelv()].`
- Ennestípus  
`{T1, ..., Tn}` típuskifejezés, ha `T1, ..., Tn` típuskifejezések  
`% pl. {'Diák Detti', [{khf, [cekla, prolog, prolog]}]}` :  
`% @type hallgato() = {atom(), [{atom(), munka()}]}`.  
`% @type munka() = nyelvlista() | integer() | ...`
- Függvénytípus  
`fun(T1, ..., Tn) -> T` típuskifejezés, ha `T1, ..., Tn` és `T` típuskifejezések

## Függvény-típus specifikálása

Egy függvény típusát az argumentumainak (formális paramétereinek) és az eredményének (visszatérési értékének) a típusa határozza meg.

- Szintaxis: `@spec funcName(T1, ..., Tn) -> Tret.`
- $T_1, \dots, T_n$  és  $T_{ret}$  háromféle lehet:
  - `TypeVar`  
Típusváltozó, tetszőleges típus jelölésére
  - `Type`  
Típuskifejezés
  - `Var :: Type`  
Paraméterváltozóval bővítve dokumentációs célra
- Paraméterváltozó: a term részeinek nevet is adhatunk, pl.:

```
% @spec safe_last(Xs::[term()]) -> {ok, X::term()} | error.  
%     X az Xs lista utolsó eleme.  
% @spec split(N::integer(), List::[term()]) ->  
%     {Prefix::[term()], Suffix::[term()]}
```

## Típus-specifikáció: példák

```
@type onOff() = on | off.
@type person() = {person, name(), age()}.
@type people() = [person()].
@type name() = {firstname, string()}.
@type age() = integer().

@spec file:open(FileName, Mode) -> {ok, Handle} | {error, Why}.
@spec file:read_line(Handle) -> {ok, Line} | eof.

@spec lists:map(fun(A) -> B, [A]) -> [B].
@spec lists:filter(fun(X) -> bool(), [X]) -> [X].

@type sspec() = {size(), board()}.
@type size() = integer().
@type board() = [[field()]].
@type field() = [info()].
@type info() = e | o | s | w | integer().
@type ssol() = [[integer()]].
@spec sudoku:sudoku(SSpec::sspec()) -> SSols::[ssol()].
```

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- **Kivételkezelés**
- Rekord
- Gyakori könyvtári függvények

# Kivételkezelés

- Kivétel jelzése háromféle *kivételtípussal* lehetséges
  - `throw(Why)`  
Olyan hiba jelzésére, amelynek kezelése várható az alkalmazástól
  - `exit(Why)`  
A futó processz befejezésére
  - `erlang:error(Why)`  
Súlyos rendszerhiba jelzésére, amelynek kezelése nem várható az alkalmazástól
- Kivétel elkapása kétféleképpen lehetséges
  - `try ... catch` kifejezéssel
  - `catch` kifejezéssel:
    - visszaadja a keletkező kivétel termjét, vagy ha nincs hiba, a kifejezés kiértékelését
    - debughoz hasznos, könnyen felderíthető a kivétel pontos értéke

## Kivételkezelés: `try ... catch`

```
try Kifejezés [of
  Minta1 [when ŐrSz1] -> Kif1;
  ...
  Mintan [when ŐrSzn] -> Kifn]
catch
  ExTípus1: ExMinta1 [when ExŐrSz1] -> ExKif1;
  ...
  ExTípusn: ExMintan [when ExŐrSzn] -> ExKifn
[after
  AfterKif]
end
```

- Ha a Kifejezés kiértékelése sikeres, az értékét az Erlang megpróbálja az `of` és `catch` közötti mintákra illeszteni
- Ha a kiértékelés sikertelen, az Erlang a jelzett kivételt próbálja meg illeszteni a `catch` és `after` közötti mintákra
- Minden esetben kiértékeli az `after` és `end` közötti kifejezést
- A `try` szerkezet speciális esete a `case`, amelyben nincs kivételkezelés

## Példák try ... catch és catch használatára

### kiv.erl – Példák kivételkezelésre

```
% Ha Fun(Arg) hibát ad, 'error', különben {ok, Fun(Arg)}.  
safe_apply(Fun, Arg) -> try Fun(Arg) of  
    V -> {ok,V}  
catch    throw:_Why -> error;  
        error:_Why -> error  
end. % például error:function_clause
```

```
2> lists:last([a,b,c]).  
c  
3> lists:last([]).  
** exception error: no function clause matching lists:last([])  
4> catch lists:last([]).  
{'EXIT',{function_clause,[...% stack trace]}}  
5> kiv:safe_apply(fun lists:last/1, [a,b,c]).  
{ok,c}  
6> kiv:safe_apply(fun lists:last/1, []).  
error
```

## Példa try ... catch és catch használatára

### kiv.erl – folytatás

```
genExc(A,1) -> A;  
genExc(A,2) -> throw(A);  
genExc(A,3) -> exit(A);  
genExc(A,4) -> erlang:error(A).
```

```
tryGenExc(X,I) -> try genExc(X,I) of  
    Val -> {I, 'Lefutott', Val}  
catch  
    throw:X -> {I, 'Kivetelt dobott', X};  
    exit:X -> {I, 'Befejezodott', X};  
    error:X -> {I, 'Sulyos hibát jelzett', X}  
end.
```

```
7> [kiv:tryGenExc(X,I) || {X,I} <- [{'No',1},{'Th',2},{'Ex',3},{'Er',4}]].  
[{1,'Lefutott','No'}, {2,'Kivetelt dobott','Th'}, {3,'Befejezodott','Ex'},  
 {4,'Sulyos hibát jelzett','Er'}]  
8> [catch kiv:genExc(X,I) || {X,I}<-[{'No',1},{'Th',2},{'Ex',3},{'Er',4}]].  
['No','Th', {'EXIT','Ex'}, {'EXIT',{'Er',[% stack trace]}]]
```



# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- Kivételkezelés
- **Rekord**
- Gyakori könyvtári függvények

# Modul

- Modul: attribútumok és függvénydeklarációk sorozata

- Attribútumok

```
-module(modname).  
-export([f1/arity1,...]).  
-import(modname,[f1/arity1,...])  
  
-compile(Opciók).  
-include("filename.hrl").  
-define(makrónév,helyettesítés).  
-undef,-ifdef,-ifndef,-else,-endif  
-vsn(verzióleírás).  
-saját_attribútum(info).
```

Modulnév: atom

Kívülről is látható függvények listája

Más modulok modulnév nélkül használható függvényeinek listája

Fordítási opciók

Rekord definíciós fájl beemelése

Makró manipuláció

Feltételes fordítás

Verzióinfó

Bővíthető saját attribútummal

- Modulinformációk lekérdezése: `m(modname)`, `modname:module_info()`

```
2> m(khf).
```

```
...
```

```
3> khf:module_info().
```

```
...
```

# Rekord

- Ha egy ennesnek sok a tagja, nehéz felidézni, melyik tag mit jelent
- Ezért vezették be a rekordot - bár önálló rekordtípus nincs
- Rekord = címkézett ennes; szintaktikai édesítőszer
- $n$  mezőjű rekord =  $n + 1$  elemű ennes:  $\{\text{rekordnév}, m_1, \dots, m_n\}$
- Rekord deklarálása (csak modulban!):  
- `record(rn, {p1=d1, ..., pn=d1}),`  
ahol
  - `rn`: rekordnév,
  - `pj`: mezőnév,
  - `dj`: alapértelmezett érték (opcionális).
- Rekord létrehozása és változóhoz kötése:  
`X=#rn{m1=v1, ..., mn=vn}`
- Egy mezőérték lekérdezése: `X#rn.mj`
- Egy/több mezőérték változóhoz kötése: `#rn{m2=V, m4=W} = X`

## Rekord: példák

A `todo.hrl` rekorddefiníciós fájl tartalma:

```
-record(todo, {sts=remind, who='HP', txt}).
```

- Csak így használhatja több Erlang modul ugyanazt a rekorddefiníciót  
(`-include("todo.hrl").` attribútum)
- Deklaráció beolvasása

```
1> rr("todo.hrl").  
[todo]
```
- Új, alapértelmezett rekord (X) létrehozása

```
2> X = #todo{}.  
#todo{sts = remind, who = 'HP', txt = undefined}
```
- X1 is új

```
3> X1 = #todo{sts=urgent, who='KR', txt="Fóliák!"}.  
#todo{sts = urgent, who = 'KR', txt = "Fóliák!"}
```
- Rekord (X1) másolása frissítéssel; X2 is új

```
4> X2 = X1#todo{sts=done}.  
#todo{sts = done, who = 'KR', txt = "Fóliák!"}
```

## Rekord: további példák

- Mezőértékek lekérdezése

```
5> #todo{who=W,txt=T} = X2.
```

```
#todo{sts = done,who = 'KR',txt = "Fóliák!"}
```

```
6> W.
```

```
'KR'
```

```
7> T.
```

```
"Fóliák!"
```

```
>8 X1#todo.sts.
```

```
urgent
```

- Rekorddeklaráció elfelejtetése

```
9> rf(todo).
```

```
ok
```

```
10> X2.
```

```
{todo,done,'KR',"Diák!"}
```

A rekord az Erlangon belül: ennes.

# Tartalom

4

## Erlang alapok

- Bevezetés
- Típusok
- Az Erlang nyelv szintaxisának alapjai
- Mintaillesztés
- Magasabbrendű függvények, függvényérték
- Listanézet
- Műveletek, beépített függvények
- Ör
- Típusspecifikáció
- Kivételkezelés
- Rekord
- **Gyakori könyvtári függvények**

## Füzérkezelő függvények (string modul)

- `len(Str)`, `equal(Str1,Str2)`, `concat(Str1,Str2)`
- `chr(Str,Chr)`, `rchr(Str,Chr)`, `str(Str,SubStr)`, `rstr(Str,SubStr)`  
A karakter / részfüzér első / utolsó előfordulásának indexe, vagy 0, ha nincs benne
- `span(Str,Chrs)`, `cspan(Str,Chrs)`  
Az `Str` ama prefixumának hossza, amelyben kizárólag a `Chars`-beli karakterek fordulnak / nem fordulnak elő
- `substr(Str,Strt,Len)`, `substr(Str,Strt)`  
Az `Str` specifikált részfüzére

## További füzérkezelő függvények (string modul)

- `tokens(Str, SepList)`  
A `SepList` karakterei mentén füzérek listájára bontja az `Str`-t
- `join(StrList, Sep)`  
Füzérré fűzi össze, `Sep`-vel elválasztva, az `StrList` elemeit
- `strip(Str)`, `strip(Str, Dir)`, `strip(Str, Dir, Char)`  
A formázó / `Char` karaktereket levágja a füzér elejéről / végéről

Részletek és továbbiak: Reference Manual.



## Listakezelő függvények (`lists` modul)

- `nth(N,Lst)`, `nthtail(N,Lst)`, `last(Lst)`  
A `Lst` `N`-edik karaktere / ott kezdődő farka / utolsó eleme
- `append(Lst1,Lst2) (++)`, `append(LstOfLsts)`  
Az `Lst1` és `Lst2` / `LstOfLsts` elemei egy listába fűzve
- `concat(Lst)`  
Az `Lst` összes eleme füzérré alakítva és egybefűzve
- `reverse(Lst)`, `reverse(Lst,T1)`  
Az `Lst` megfordítva / megfordítva a `T1` elé fűzve (más deklaratív nyelvekben `reverse/2`-nek `revAppend` a neve)
- `flatten(DeepList)`, `flatten(DeepList,Tail)`  
A `DeepList` kisimítva / kisimítva `Tail` elé fűzve
- `max(Lst)`, `min(Lst)`  
Az `Lst` legnagyobb / legkisebb eleme

## További listakezelő függvények (`lists` modul)

- `filter(Pred,Lst)`, `delete(Elem,Lst)`  
A `Lst` `Pred`-et kielégítő elemek / `Elem` nélküli másolata
- `takewhile(Pred,Lst)`, `dropwhile(Pred,Lst)`,  
`splitwith(Pred,Lst)`  
Az `Lst` `Pred`-et kielégítő prefixumát tartalmazó / nem tartalmazó másolata; ilyen listákból álló pár
- `partition(Pred,Lst)`, `split(N,Lst)`  
A `Lst` elemei `Pred` / `N` szerint két listába válogatva
- `member(Elem,Lst)`, `all(Pred,Lst)`, `any(Pred,Lst)`  
Igaz, ha `Elem` / `Pred` szerinti minden / `Pred` szerinti legalább egy elem benne van az `Lst`-ben
- `prefix(Lst1,Lst2)`, `suffix(Lst1,Lst2)`  
Igaz, ha az `Lst2` az `Lst1`-gyel kezdődik / végződik

## Továbbra is: listakezelő függvények (`lists` modul)

- `sublist(Lst,Len)`, `sublist(Lst,Strt,Len)`  
Az `Lst` 1-től / `Strt`-től kezdődő, `Len` hosszú része
- `subtract(Lst1,Lst2) (--)`  
Az `Lst1` `Lst2` elemeinek első előfordulását nem tartalmazó másolata
- `zip(Lst1,Lst2)`, `unzip(Lst)`  
Az `Lst1` és `Lst2` elemeiből képzett párok listája; az `Lst`-ben lévő párok szétválasztásával létrehozott két lista
- `sort(Lst)`, `sort(Fun,Lst)`  
Az `Lst` alapértelmezés / `Fun` szerint rendezett másolata
- `merge(LstOfLsts)`  
Az `LstOfLsts` listában lévő rendezett listák alapértelmezés szerinti összefuttatása

## Még mindig: listakezelő függvények (`lists` modul)

- `merge(Lst1, Lst2)`, `merge(Fun, Lst1, Lst2)`,  
A rendezett `Lst1` és `Lst2` listák alapértelmezés / `Fun` szerinti összefuttatása
- `map(Fun, Lst)`  
Az `Lst` `Fun` szerinti átalakított elemeiből álló lista
- `foreach(Fun, Lst)`  
Az `Lst` elemeire a mellékhatást okozó `Fun` alkalmazása
- `sum(Lst)`  
Az `Lst` elemeinek összege, ha az összes elem számot eredményező kifejezés
- `foldl(Fun, Acc, Lst)`, `foldr(Fun, Acc, Lst)`  
Az `Acc` akkumulátor és az `Lst` elemeinek `Fun` szerinti redukálása, balról jobbra, illetve jobbról balra haladva

Részletek és továbbiak: Reference Manual.

## Néhány további könyvtári modul és függvény

- **math modul:** `pi()`, `sin(X)`, `acos(X)`, `tanh(X)`, `asinh(X)`, `exp(X)`, `log(X)`, `log10(X)`, `pow(X,Y)`, `sqrt(X)`
- **io modul:** `write([IoDev,]Term)`, `fwrite(Format)`, `fwrite([IoDev,]Format,Data)`, `nl([IoDev])`, `format(Format)`, `format([IoDev,]Format,Data)`, `get_line([IoDev,]Prompt)`, `read([IoDev,]Prompt)`
- **Formázójelek (io modul)**

|        |         |            |                        |
|--------|---------|------------|------------------------|
| ~~     | a ~ jel | ~c         | az adott kódú karakter |
| ~s     | fűzér   | ~f, ~e, ~g | lebegőpontos szám      |
| ~b, ~x | egész   | ~w, ~p     | Erlang-term            |
| ~n     | újsor   |            |                        |

```
1> io:format("~s ~b ~c ~f~n", [[a,$b,$c],$a,$b,math:exp(1)]).
```

```
abc 97 b 2.718282
```

```
ok
```

```
3> X={"abc", [1,2,3], at}, io:format(" p w n", [X,X]).
```

```
{"abc", [1,2,3], at} {[97,98,99], [1,2,3], at}
```

```
ok
```