Deklaratív programozás 2. előadás

Kabódi László[1] Marussy Kristóf[2]
BME Számítástudományi és Információelméleti Tanszék
Mesterséges Intelligencia és Rendszertervezés Tanszék

[1] kabodi.laszlo@vik.bme.hu
[2] marussy@mit.bme.hu

2026. ősz

I. rész

Típusok, termek, azonosítók, változók

Tartalom

Típusok[1]

Az Elixir erősen típusos nyelv, dinamikus típusellenőrzéssel.

Értéktípusok	**Value types**
Atom	Atom
Tetszőleges hosszú egész szám	Arbitrary-sized integer (integer)
Lebegőpontos szám	Floating-point number (float)
Függvény	Function
Tartomány	*Range*
Reguláris kifejezés	*Regular expression (regex)*
Sztring	*String*

Kollekció-típusok	**Collection types**
Ennes	Tuple
Lista	List
Bináris	Binary
Szótár	Map
Struktúra	Struct

[1] A felsorolás nem teljes. A dőlt betűs típusok más alaptípusokra épülnek.

Atom

- Kettősponttal (:) kezdődik
- Kezdődhet az angol ábécé nagybetűjével is, kettőspont nélkül, de ez konvenció szerint a modulnevekre van fenntartva
- A : után UTF-8 kódolású karaktersorozat, Elixir operátor vagy sztring állhat
- Az UTF-8 kódolású karaktersorozatban betűk, számjegyek és kétféle írásjel (_, @) lehetnek
- A karaktersorozat végén általában kérdőjel (?) vagy felkiáltójel (!) is lehet
- Saját magát jelöli, nem sztring: egy atom értéke maga a neve
- Két azonos nevű atom mindig egyenlő, akárhol is vannak definiálva
- Hasonló a Prolog névkonstanshoz (atomhoz)
- Példák: `:János`, `:is_bin?`, `:vált@2`, `:<>`, `:"fun/3"`, `:"éljen soká!"`, `:Éljen_soká!`, `:"Őrült Űrőr tűrjön"`, `Dp`, `Gy1`

Szám

- Egész (integer)
  - Decimális, pl. 1234
  - Hexadecimális, pl. 0xcafe
  - Oktális, pl. 0o765
  - Bináris, pl. 0b1010
  - Tagolható, pl. `123_456_789`
  - Korlátlan pontosságú, pl. `123456789012345678901234567890`
  - Karakterkód (Unicode codepoint)
    - Ha nyomtatható: `?z`
    - Ha vezérlő: `?\n`

Szám

- Egész (integer)
  - Decimális, pl. 1234
  - Hexadecimális, pl. 0xcafe
  - Oktális, pl. 0o765
  - Bináris, pl. 0b1010
  - Tagolható, pl. `123_456_789`
  - Korlátlan pontosságú, pl. `123456789012345678901234567890`
  - Karakterkód (Unicode codepoint)
    - Ha nyomtatható: `?z`
    - Ha vezérlő: `?\n`
- Lebegőpontos (float)
  - Pl. `3.14159`,
  - Vezető nullával, pl. `0.14159`
  - Exponenssel pl. `0.2e-22`
  - IEEE 754 szerinti, dupla pontosságú
    (64 bit, kb. 16 számjegy, max. exponens kb. 10^{308})

Függvény (Function) 1

- A függvény is érték: változóhoz köthető, adatstruktúra eleme lehet, függvény eredménye lehet, paraméterként átadható stb.

  Azaz: a függvény is ún. *first class citizen, teljes jogú polgár*

Függvény (Function) 1

- A függvény is érték: változóhoz köthető, adatstruktúra eleme lehet, függvény eredménye lehet, paraméterként átadható stb.

  Azaz: a függvény is ún. *first class citizen, teljes jogú polgár*
- Példák:

```
iex> fac = &Fpea.fac/1 # &: capture operator
&Fpea.fac/1
iex>
```

Függvény (Function) 1

- A függvény is érték: változóhoz köthető, adatstruktúra eleme lehet, függvény eredménye lehet, paraméterként átadható stb.

  Azaz: a függvény is ún. *first class citizen, teljes jogú polgár*
- Példák:

```
iex> fac = &Fpea.fac/1 # &: capture operator
&Fpea.fac/1
iex> fac.(5) # pont és zárójelpár kell, szóközökkel tagolható
120
iex> Kernel.+(3,2) # infix operátor alkalmazása prefix helyzetben
5
iex>
```

Függvény (Function) 1

- A függvény is érték: változóhoz köthető, adatstruktúra eleme lehet, függvény eredménye lehet, paraméterként átadható stb.

  Azaz: a függvény is ún. *first class citizen, teljes jogú polgár*
- Példák:

```
iex> fac = &Fpea.fac/1 # &: capture operator
&Fpea.fac/1
iex> fac.(5) # pont és zárójelpár kell, szóközökkel tagolható
120
iex> Kernel.+(3,2) # infix operátor alkalmazása prefix helyzetben
5
iex> fs = [&Kernel.+/2, &*/2, &:math.sin/1] # :math Erlang modul!
[&:erlang.+/2, &:erlang.*/2, &:math.sin/1]
iex> (hd fs).(3,2)
5
iex> (hd tl fs).(3,2)
6
iex> (hd tl tl fs).(:math.pi * 90 / 180)
1.0
```

Függvény (Function) 2

- További példák: anonim függvény definiálása, hívása, névhez kötése

```
iex> fn ki -> "Szia, " <> ki <> "!" end # <>: konkatenálás
#Function<44.40011524/1 in :erl_eval.expr/5>
iex>
```

Függvény (Function) 2

- További példák: anonim függvény definiálása, hívása, névhez kötése

```
iex> fn ki -> "Szia, " <> ki <> "!" end # <>: konkatenálás
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> fn ki -> "Szia, "<>ki<>"!" end.("Péter") # pont, zárójel!
"Szia, Péter!"
iex>
```

Függvény (Function) 2

- További példák: anonim függvény definiálása, hívása, névhez kötése

```
iex> fn ki -> "Szia, " <> ki <> "!" end # <>: konkatenálás
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> fn ki -> "Szia, "<>ki<>"!" end.("Péter") # pont, zárójel!
"Szia, Péter!"
iex> szia = fn ki -> "Szia, " <> ki <> "!" end
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> szia
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> szia.("Bea")
"Szia, Bea!"
```

Függvény (Function) 2

- További példák: anonim függvény definiálása, hívása, névhez kötése

```
iex> fn ki -> "Szia, " <> ki <> "!" end # <>: konkatenálás
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> fn ki -> "Szia, "<>ki<>"!" end.("Péter") # pont, zárójel!
"Szia, Péter!"
iex> szia = fn ki -> "Szia, " <> ki <> "!" end
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> szia
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> szia.("Bea")
"Szia, Bea!"
```

- További példa: függvénydefiníció `def`-fel, `defp`-vel

```
def sum_of_squares(a,b), do: sqr(a) + sqr(b)
defp sqr(a), do: a*a # p[rivát], azaz lokális a modulon belül
iex> Fpea.sum_of_squares 3, 4.5
29.25
```

Függvény (Function) 2

- További példák: anonim függvény definiálása, hívása, névhez kötése

```
iex> fn ki -> "Szia, " <> ki <> "!" end # <>: konkatenálás
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> fn ki -> "Szia, "<>ki<>"!" end.("Péter") # pont, zárójel!
"Szia, Péter!"
iex> szia = fn ki -> "Szia, " <> ki <> "!" end
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> szia
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> szia.("Bea")
"Szia, Bea!"
```

- További példa: függvénydefiníció `def`-fel, `defp`-vel

```
def sum_of_squares(a,b), do: sqr(a) + sqr(b)
defp sqr(a), do: a*a # p[rivát], azaz lokális a modulon belül
iex> Fpea.sum_of_squares 3, 4.5
29.25
```

- Függvény *típusa:* `(`*arg1 típusa*`,` *arg2 típusa*`, ...) ::` *eredmény típusa*
  Pl. a `sum_of_squares/2` függvényé: `(number, number) :: number`

Paraméter alapértelmezett (default) értéke

- Egy függvény egy vagy több paraméterének adhatunk alapértelmezett értéket a \\ jelöléssel. Az ilyen paraméter opcionális, a többi elvárt.
- Ha egy függvényt
  - a kötelezően (default argumentumok nélkül) elvártnál kevesebb paraméterrel hívunk meg, a hívás meghiúsul;
  - az elvárt számú paraméterrel hívunk meg, az összes opcionális paraméter az alapértelmezett értékét veszi fel;
  - az elvártnál több paraméterrel hívunk meg, az aktuális paraméterek értékét balról jobbra haladva veszik fel az opcionális paraméterek.

Paraméter alapértelmezett (default) értéke

- Egy függvény egy vagy több paraméterének adhatunk alapértelmezett értéket a `\\` jelöléssel. Az ilyen paraméter opcionális, a többi elvárt.
- Ha egy függvényt
  - a kötelezően (default argumentumok nélkül) elvártnál kevesebb paraméterrel hívunk meg, a hívás meghiúsul;
  - az elvárt számú paraméterrel hívunk meg, az összes opcionális paraméter az alapértelmezett értékét veszi fel;
  - az elvártnál több paraméterrel hívunk meg, az aktuális paraméterek értékét balról jobbra haladva veszik fel az opcionális paraméterek.
- Példák alapértelmezett értékekkel

```
def sum_of_sqrs_b5(a, b \\ 5), do: sqr(a) + sqr(b)
iex> Fpea.sum_of_sqrs_b5 3, 4.5
29.25
iex> Fpea.sum_of_sqrs_b5 3
34
```

Paraméter alapértelmezett (default) értéke

- Egy függvény egy vagy több paraméterének adhatunk alapértelmezett értéket a \\ jelöléssel. Az ilyen paraméter opcionális, a többi elvárt.
- Ha egy függvényt
  - a kötelezően (default argumentumok nélkül) elvártnál kevesebb paraméterrel hívunk meg, a hívás meghiúsul;
  - az elvárt számú paraméterrel hívunk meg, az összes opcionális paraméter az alapértelmezett értékét veszi fel;
  - az elvártnál több paraméterrel hívunk meg, az aktuális paraméterek értékét balról jobbra haladva veszik fel az opcionális paraméterek.
- Példák alapértelmezett értékekkel

```
def sum_of_sqrs_b5(a, b \\ 5), do: sqr(a) + sqr(b)
iex> Fpea.sum_of_sqrs_b5 3, 4.5
29.25
iex> Fpea.sum_of_sqrs_b5 3
34
def sum_of_sqrs_b5(a \\ 6, b \\ 5), do: sqr(a) + sqr(b)
iex> Fpea.sum_of_sqrs_a6b5 3
34
iex> Fpea.sum_of_sqrs_a6b5
61
```

Ennes (Tuple), tartomány (Range)

Ennes (Tuple)

- Rögzített számú, tetszőleges kifejezésből álló, fix sorrendű kollekció“
- Példák:

```
iex> {0x1ff, :erlang, Armstrong, 'Joe'++[0], [], {}}
{511, :erlang, Armstrong, [74, 111, 101, 0], [], {}}
iex> {plus, per, sin} = # mintaillesztések kötésekkel
  {&Kernel.+/2, &//2, &:math.sin/1}
{&:erlang.+/2, &:erlang.//2, &:math.sin/1}
iex> {plus.(3,4), per.(3,4)} # infix volt, prefix lett
{7, 0.75}
iex> sin.(90*:math.pi/180)
1.0
```

Ennes (Tuple), tartomány (Range)

Ennes (Tuple)

- Rögzített számú, tetszőleges kifejezésből álló, fix sorrendű kollekció“
- Példák:

```
iex> {0x1ff, :erlang, Armstrong, 'Joe'++[0], [], {}}
{511, :erlang, Armstrong, [74, 111, 101, 0], [], {}}
iex> {plus, per, sin} = # mintaillesztések kötésekkel
  {&Kernel.+/2, &//2, &:math.sin/1}
{&:erlang.+/2, &:erlang.//2, &:math.sin/1}
iex> {plus.(3,4), per.(3,4)} # infix volt, prefix lett
{7, 0.75}
iex> sin.(90*:math.pi/180)
1.0
```

Tartomány (Range)

- Egész számok sorozata a *[start, end]* tartományban
- Példa tartomány és lépésköz definiálására, használatára:

```
iex> {18..23, 18..10}
{18..23, 18..10//-1}
iex> for i <- 18..10 // -3, do: i
[18, 15, 12]
```

Lista (List)

- Korlátlan számú, tetszőleges kifejezésből álló, *egyszeresen láncolt* sorozat
- Lineáris rekurzív adatstruktúra:
  - vagy üres (`[]` jellel jelöljük),
  - vagy egy elemből áll, amelyet egy lista követ: `[x|xs]`
- Első eleme, ha van, a lista *feje*
- Első eleme utáni, esetleg üres része a lista *farka*

Lista (List)

- Korlátlan számú, tetszőleges kifejezésből álló, *egyszeresen láncolt* sorozat
- Lineáris rekurzív adatstruktúra:
  - vagy üres (`[]` jellel jelöljük),
  - vagy egy elemből áll, amelyet egy lista követ: `[x|xs]`
- Első eleme, ha van, a lista *feje*
- Első eleme utáni, esetleg üres része a lista *farka*

```
iex> [:elem] # egyelemű lista
[:elem]
iex> [:elem|[]] # fejből és üres farokból létrehozott lista
[:elem]
iex> [:elem1|[:elem2]] # fejből-farokból létrehozott lista
[:elem1, :elem2]
iex> [:elem,123,3.14,'elem'] # több elemű listák
[:elem, 123, 3.14, 'elem']
iex> [:elem,123|[3.14,'elem']]
[:elem, 123, 3.14, 'elem']
iex> [:egy|[:két]] ++ [:elem,123|[3.14,'elem']] # ++: konkatenáció
[:egy, :két, :elem, 123, 3.14, 'elem']
```

Karakterlánc (single-quoted)

- Rövidítés, karakterkódok listája: `'erl'` ≡ `[?e,?r,?l]` ≡ `[101,114,108]`
- Az Elixir/Erlang shell a nyomtatható karakterkódok (7..13, 27, 32..126) listáját karakterláncként írja ki
- Ha ezektől különböző érték is van a listában, listaként írja ki
- Példák:

```
iex> [101,114,108]
~c"erl"
iex> [31,101,114,108]
[31, 101, 114, 108]
iex> 'erl' ++ 'ang' # konkatenálható
~c"erlang"
```

- A karakterlánc NEM sztring!

Sztring (String, double quoted)

- UTF-8 kódolású karakterek ábrázolása bájtok sorozataként (bináris típus)
- Következmények:
  - Az UTF-8 kódolás miatt a sztring rövidebb lehet az őt ábrázoló binárisnál
  - A lista- és a sztringműveletek különbözőek
- Példák:

```
iex> dxdy = "δx/δy"
"δx/δy"
iex>
```

Sztring (String, double quoted)

- UTF-8 kódolású karakterek ábrázolása bájtok sorozataként (bináris típus)
- Következmények:
  - Az UTF-8 kódolás miatt a sztring rövidebb lehet az őt ábrázoló binárisnál
  - A lista- és a sztringműveletek különbözőek
- Példák:

```
iex> dxdy = "δx/δy"
"δx/δy"
iex> {String.length(dxdy), byte_size(dxdy)}
{5, 7}
iex>
```

Sztring (String, double quoted)

- UTF-8 kódolású karakterek ábrázolása bájtok sorozataként (bináris típus)
- Következmények:
  - Az UTF-8 kódolás miatt a sztring rövidebb lehet az őt ábrázoló binárisnál
  - A lista- és a sztringműveletek különbözőek
- Példák:

```
iex> dxdy = "δx/δy"
"δx/δy"
iex> {String.length(dxdy), byte_size(dxdy)}
{5, 7}
iex> {String.at(dxdy,0), String.codepoints(dxdy)}
{"δ", ["δ", "x", "/", "δ", "y"]}
iex>
```

Sztring (String, double quoted)

- UTF-8 kódolású karakterek ábrázolása bájtok sorozataként (bináris típus)
- Következmények:
  - Az UTF-8 kódolás miatt a sztring rövidebb lehet az őt ábrázoló binárisnál
  - A lista- és a sztringműveletek különbözőek
- Példák:

```
iex> dxdy = "δx/δy"
"δx/δy"
iex> {String.length(dxdy), byte_size(dxdy)}
{5, 7}
iex> {String.at(dxdy,0), String.codepoints(dxdy)}
{"δ", ["δ", "x", "/", "δ", "y"]}
iex> [dx, dy] = String.split(dxdy, "/")
["δx", "δy"]
iex>
```

Sztring (String, double quoted)

- UTF-8 kódolású karakterek ábrázolása bájtok sorozataként (bináris típus)
- Következmények:
  - Az UTF-8 kódolás miatt a sztring rövidebb lehet az őt ábrázoló binárisnál
  - A lista- és a sztringműveletek különbözőek
- Példák:

```
iex> dxdy = "δx/δy"
"δx/δy"
iex> {String.length(dxdy), byte_size(dxdy)}
{5, 7}
iex> {String.at(dxdy,0), String.codepoints(dxdy)}
{"δ", ["δ", "x", "/", "δ", "y"]}
iex> [dx, dy] = String.split(dxdy, "/")
["δx", "δy"]
iex> dx <> "/" <> dy # <>: konkatenálás
"δx/δy"
```

Sztring (String, double quoted)

- UTF-8 kódolású karakterek ábrázolása bájtok sorozataként (bináris típus)
- Következmények:
  - Az UTF-8 kódolás miatt a sztring rövidebb lehet az őt ábrázoló binárisnál
  - A lista- és a sztringműveletek különbözőek
- Példák:

```
iex> dxdy = "δx/δy"
"δx/δy"
iex> {String.length(dxdy), byte_size(dxdy)}
{5, 7}
iex> {String.at(dxdy,0), String.codepoints(dxdy)}
{"δ", ["δ", "x", "/", "δ", "y"]}
iex> [dx, dy] = String.split(dxdy, "/")
["δx", "δy"]
iex> dx <> "/" <> dy # <>: konkatenálás
"δx/δy"
```

- Sztringműveletekről, a *String* modul függvényeiről hamarosan lesz szó

Ami közös a karakterláncban és a sztringben

- UTF-8 kódolású karakterekből állnak
- pause Lehetnek bennük ún. escape-szekvenciák:

\a	BEL (0x07)	\b	BS (0x08)	\d	DEL (0x7f)
\e	ESC (0x1b)	\f	FF (0x0c)	\n	NL (0x0a)
\r	CR (0x0d)	\s	SP (0x20)	\t	TAB (0x09)
\v	VT (0x0b)	\uhhhh	Unicode codepoint in hexadecimal		
\xhh	single byte in hexadecimal				

- Néhány karakter speciális jelentését az elé írt \ megszünteti, pl. \\

Ami közös a karakterláncban és a sztringben

- UTF-8 kódolású karakterekből állnak
- pause Lehetnek bennük ún. escape-szekvenciák:

\a	BEL (0x07)	\b	BS (0x08)	\d	DEL (0x7f)
\e	ESC (0x1b)	\f	FF (0x0c)	\n	NL (0x0a)
\r	CR (0x0d)	\s	SP (0x20)	\t	TAB (0x09)
\v	VT (0x0b)	\uhhhh	Unicode codepoint in hexadecimal		
\xhh	single byte in hexadecimal				

- Néhány karakter speciális jelentését az elé írt \ megszünteti, pl. \\
- Megengedik az ún. *interpolációt*, azaz változó helyettesítését az értékével ("...#{<expr>}...") sztringben, illetve karakterláncban:

```
iex> name = "dávid" # Sztring
"dávid"
iex> "Helló, #{String.capitalize name}!"
"Helló, Dávid!"
iex> bubo = 'Bubo' # Karakterlánc
~c"Bubo"
iex> "Helló, #{List.to_string [bubo, ? , "Réka"]}!"
"Helló, Bubo Réka!"
```

Bináris (Binary)

- A bináris típusba tartozó értékek bitsorozatok
- Egy bináris érték jelölése `<< kif, ... >>` alakú
- A legegyszerűbb `kif` a [0,255] tartományba eső egész szám
- A számokat bájtként tároljuk a binárisban

```
iex> b = << 1, 2, 3 >>
<<1, 2, 3>>
iex> {byte_size(b), bit_size b}
{3, 24}
```

Bináris (Binary)

- A bináris típusba tartozó értékek bitsorozatok
- Egy bináris érték jelölése `<< kif, ... >>` alakú
- A legegyszerűbb `kif` a [0,255] tartományba eső egész szám
- A számokat bájtként tároljuk a binárisban

```
iex> b = << 1, 2, 3 >>
<<1, 2, 3>>
iex> {byte_size(b), bit_size b}
{3, 24}
```

- A tárolásra használt bitek száma megszabható

```
iex> b = << 1::size(2), 1::size(3) >> # 01 001
<<9::size(5)>> # = 9 (decimálisként)
iex> {byte_size(b), bit_size b}
{1, 5}
```

Bináris (Binary)

- A bináris típusba tartozó értékek bitsorozatok
- Egy bináris érték jelölése `<< kif, ... >>` alakú
- A legegyszerűbb `kif` a [0,255] tartományba eső egész szám
- A számokat bájtként tároljuk a binárisban

```
iex> b = << 1, 2, 3 >>
<<1, 2, 3>>
iex> {byte_size(b), bit_size b}
{3, 24}
```

- A tárolásra használt bitek száma megszabható

```
iex> b = << 1::size(2), 1::size(3) >> # 01 001
<<9::size(5)>> # = 9 (decimálisként)
iex> {byte_size(b), bit_size b}
{1, 5}
```

- Egész és lebegőpontos számok és más értékek is tárolhatók binárisan

```
iex> << <<1>> :: binary, <<2.5>> :: binary >>
<<1, 64, 4, 0, 0, 0, 0, 0, 0>>
```

- A bináris tárolás hasznos médiafájlok és UTF-8 karakterek tárolására, processzek közötti kommunikációban stb.

Kulcs-érték lista (Keyword lists)

- Egy kulcs-érték párt kételemű enneskénti írhatunk le: `{:key, value}`, ahol a kulcs csak atom, az érték tetszőleges típusú lehet
- Gyakran van szükség ilyen listákra, ezért az Elixir többféle jelölést, rövidítést, bizonyos esetekben zárójelelhagyást is megenged

Kulcs-érték lista (Keyword lists)

- Egy kulcs-érték párt kételemű enneskénti írhatunk le: `{:key, value}`, ahol a kulcs csak atom, az érték tetszőleges típusú lehet
- Gyakran van szükség ilyen listákra, ezért az Elixir többféle jelölést, rövidítést, bizonyos esetekben zárójelelhagyást is megenged
- Példák

```
iex> [{:név,"Szöszi"},{:szerelme,"jazz-zongorista"},{:város,"Prága"}]
[név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"]
iex>
```

Kulcs-érték lista (Keyword lists)

- Egy kulcs-érték párt kételemű enneskénti írhatunk le: `{:key, value}`, ahol a kulcs csak atom, az érték tetszőleges típusú lehet
- Gyakran van szükség ilyen listákra, ezért az Elixir többféle jelölést, rövidítést, bizonyos esetekben zárójelelhagyást is megenged
- Példák

```
iex> [{:név,"Szöszi"},{:szerelme,"jazz-zongorista"},{:város,"Prága"}]
[név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"]
iex> [név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"]
[név: "Szöszi", szerelme: "jazz-zongorista", "város: "Prága"]
iex>
```

Kulcs-érték lista (Keyword lists)

- Egy kulcs-érték párt kételemű enneskénti írhatunk le: `{:key, value}`, ahol a kulcs csak atom, az érték tetszőleges típusú lehet
- Gyakran van szükség ilyen listákra, ezért az Elixir többféle jelölést, rövidítést, bizonyos esetekben zárójelelhagyást is megenged
- Példák

```
iex> [{:név,"Szöszi"},{:szerelme,"jazz-zongorista"},{:város,"Prága"}]
[név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"]
iex> [név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"]
[név: "Szöszi", szerelme: "jazz-zongorista", "város: "Prága"]
iex> inspect név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"
"[név: \"Szöszi\", szerelme: \"jazz-zongorista\", város: \"Prága\"]"
iex>
```

Kulcs-érték lista (Keyword lists)

- Egy kulcs-érték párt kételemű enneskénti írhatunk le: `{:key, value}`, ahol a kulcs csak atom, az érték tetszőleges típusú lehet
- Gyakran van szükség ilyen listákra, ezért az Elixir többféle jelölést, rövidítést, bizonyos esetekben zárójelelhagyást is megenged
- Példák

```
iex> [{:név,"Szöszi"},{:szerelme,"jazz-zongorista"},{:város,"Prága"}]
[név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"]
iex> [név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"]
[név: "Szöszi", szerelme: "jazz-zongorista", "város: "Prága"]
iex> inspect név: "Szöszi", szerelme: "jazz-zongorista", város: "Prága"
"[név: \"Szöszi\", szerelme: \"jazz-zongorista\", város: \"Prága\"]"
iex> [:cseh_film, név: "Szöszi", város: "Prága", szerelme: "zongorista"]
[:cseh_film, {név: "Szöszi", város: "Prága", szerelme: "zongorista"}]
iex> {:cseh_film, név: "Szöszi", szerelme: "zongorista", város: "Prága"}
{:cseh_film, [név: "Szöszi", szerelme: "zongorista", város: "Prága"]}
```

- A kulcs-érték párokat leginkább függvényopciók megadására használjuk, pl. `limit: :infinity`, `charlists: :as_lists`

Szótár (Map) 1

- A szótár kulcs-érték párok rendezett kollekciója
- Jelölés (map literal): `%{ key1 => value1, key2 => value2, ...}`
- Ha a kulcs atom, alternatív jelölés: `%{atom1: value1, atom2: value2}`
- A kulcsok és az értékek típusa tetszőleges; lehet kifejezés is
- Egy szótáron belül a kulcsok különböző típusúak lehetnek

Szótár (Map) 1

- A szótár kulcs-érték párok rendezett kollekciója
- Jelölés (map literal): `%{ key1 => value1, key2 => value2, ...}`
- Ha a kulcs atom, alternatív jelölés: `%{atom1: value1, atom2: value2}`
- A kulcsok és az értékek típusa tetszőleges; lehet kifejezés is
- Egy szótáron belül a kulcsok különböző típusúak lehetnek
- Példák:

```
iex> states = %{"UA"=>"Ukraine", "SK"=>"Slovakia", "AT"=>"Austria"}
%{"AT" => "Austria", "SK" => "Slovakia", "UA" => "Ukraine"}
iex> msgs = %{{:error,:enoent} => :fatal, {:error,:busy} => :retry}
%{{:error, :busy} => :retry, {:error, :enoent} => :fatal}
iex> colors = %{:red=>0xff0000, :green=>0x00ff00, :blue=>0x0000ff}
%{green: 65280, red: 16711680, blue: 255}
iex>
```

Szótár (Map) 1

- A szótár kulcs-érték párok rendezett kollekciója
- Jelölés (map literal): `%{ key1 => value1, key2 => value2, ...}`
- Ha a kulcs atom, alternatív jelölés: `%{atom1: value1, atom2: value2}`
- A kulcsok és az értékek típusa tetszőleges; lehet kifejezés is
- Egy szótáron belül a kulcsok különböző típusúak lehetnek
- Példák:

```
iex> states = %{"UA"=>"Ukraine", "SK"=>"Slovakia", "AT"=>"Austria"}
%{"AT" => "Austria", "SK" => "Slovakia", "UA" => "Ukraine"}
iex> msgs = %{{:error,:enoent} => :fatal, {:error,:busy} => :retry}
%{{:error, :busy} => :retry, {:error, :enoent} => :fatal}
iex> colors = %{:red=>0xff0000, :green=>0x00ff00, :blue=>0x0000ff}
%{green: 65280, red: 16711680, blue: 255}
iex> colors = %{red: 0xff0000, green: 0x00ff00, blue: 0x0000ff}
%{green: 65280, red: 16711680, blue: 255}
iex>
```

Szótár (Map) 1

- A szótár kulcs-érték párok rendezett kollekciója
- Jelölés (map literal): `%{ key1 => value1, key2 => value2, ...}`
- Ha a kulcs atom, alternatív jelölés: `%{atom1: value1, atom2: value2}`
- A kulcsok és az értékek típusa tetszőleges; lehet kifejezés is
- Egy szótáron belül a kulcsok különböző típusúak lehetnek
- Példák:

```
iex> states = %{"UA"=>"Ukraine", "SK"=>"Slovakia", "AT"=>"Austria"}
%{"AT" => "Austria", "SK" => "Slovakia", "UA" => "Ukraine"}
iex> msgs = %{{:error,:enoent} => :fatal, {:error,:busy} => :retry}
%{{:error, :busy} => :retry, {:error, :enoent} => :fatal}
iex> colors = %{:red=>0xff0000, :green=>0x00ff00, :blue=>0x0000ff}
%{green: 65280, red: 16711680, blue: 255}
iex> colors = %{red: 0xff0000, green: 0x00ff00, blue: 0x0000ff}
%{green: 65280, red: 16711680, blue: 255}
iex> mix = %{(&+/2).(3,2) => "három+kettő", fütty: "dal"<>"olka"}
%{5 => "három+kettő", :fütty => "dalolka"}
```

Szótár (Map) 2

- A szótár típust elsősorban asszociatív tömbként szokás használni
- Szótárból értéket a kulccsal lehet kinyerni szögletes zárójeles jelöléssel
- Ha a kulcs atom, a rövidebb pontos jelölés is használható

Szótár (Map) 2

- A szótár típust elsősorban asszociatív tömbként szokás használni
- Szótárból értéket a kulccsal lehet kinyerni szögletes zárójeles jelöléssel
- Ha a kulcs atom, a rövidebb pontos jelölés is használható
- Példák:

```
iex> states["UA"]
"Ukraine"
iex>
```

Szótár (Map) 2

- A szótár típust elsősorban asszociatív tömbként szokás használni
- Szótárból értéket a kulccsal lehet kinyerni szögletes zárójeles jelöléssel
- Ha a kulcs atom, a rövidebb pontos jelölés is használható
- Példák:

```
iex> states["UA"]
"Ukraine"
iex> states["HU"]
nil
iex>
```

Szótár (Map) 2

- A szótár típust elsősorban asszociatív tömbként szokás használni
- Szótárból értéket a kulccsal lehet kinyerni szögletes zárójeles jelöléssel
- Ha a kulcs atom, a rövidebb pontos jelölés is használható
- Példák:

```
iex> states["UA"]
"Ukraine"
iex> states["HU"]
nil
iex> msgs[{:error, :busy}]
:retry
iex> colors[:green]
65280
iex>
```

Szótár (Map) 2

- A szótár típust elsősorban asszociatív tömbként szokás használni
- Szótárból értéket a kulccsal lehet kinyerni szögletes zárójeles jelöléssel
- Ha a kulcs atom, a rövidebb pontos jelölés is használható
- Példák:

```
iex> states["UA"]
"Ukraine"
iex> states["HU"]
nil
iex> msgs[{:error, :busy}]
:retry
iex> colors[:green]
65280
iex> colors.red
16711680
iex>
```

Szótár (Map) 2

- A szótár típust elsősorban asszociatív tömbként szokás használni
- Szótárból értéket a kulccsal lehet kinyerni szögletes zárójeles jelöléssel
- Ha a kulcs atom, a rövidebb pontos jelölés is használható
- Példák:

```
iex> states["UA"]
"Ukraine"
iex> states["HU"]
nil
iex> msgs[{:error, :busy}]
:retry
iex> colors[:green]
65280
iex> colors.red
16711680
iex> mix[(&Kernel.*/2).(1,5)]
"három+kettő"
iex> mix.fütty
"dalolka"
```

- További részletek a *Map* modul dokumentációjában

Reguláris kifejezés (Regex) 1

- Az Elixirben a reguláris kifejezés is önálló típus
- Jelölés: `~r{regexp}`[2] vagy `~r{regexp}options`
- A reguláris kifejezés szintaxisa a PCRE[3] szerinti.

[2] A `~r{...}` jelölés is egy *szigil*, azaz bűvös jelölés. A szigilekről részletek a Kernel dokumentációjában találhatók.

[3] Perl Compatible Regular Expressions, `http://www.pcre.org`

Reguláris kifejezés (Regex) 1

- Az Elixirben a reguláris kifejezés is önálló típus
- Jelölés: `~r{regexp}`[2] vagy `~r{regexp}options`
- A reguláris kifejezés szintaxisa a PCRE[3] szerinti.
- Példák:

```
iex> Regex.run ~r{[cdr]}, "madárcsicsergés"
["d"]
iex>
```

[2] A `~r{...}` jelölés is egy *szigil*, azaz bűvös jelölés. A szigilekről részletek a Kernel dokumentációjában találhatók.

[3] Perl Compatible Regular Expressions, `http://www.pcre.org`

Reguláris kifejezés (Regex) 1

- Az Elixirben a reguláris kifejezés is önálló típus
- Jelölés: `~r{regexp}`[2] vagy `~r{regexp}options`
- A reguláris kifejezés szintaxisa a PCRE[3] szerinti.
- Példák:

```
iex> Regex.run ~r{[cdr]}, "madárcsicsergés"
["d"]
iex> Regex.scan ~r{[cdr]}, "madárcsicsergés"
[["d"], ["r"], ["c"], ["c"], ["r"]]
iex>
```

[2] A `~r{...}` jelölés is egy *szigil*, azaz bűvös jelölés. A szigilekről részletek a Kernel dokumentációjában találhatók.

[3] Perl Compatible Regular Expressions, `http://www.pcre.org`

Reguláris kifejezés (Regex) 1

- Az Elixirben a reguláris kifejezés is önálló típus
- Jelölés: `~r{regexp}`[2] vagy `~r{regexp}options`
- A reguláris kifejezés szintaxisa a PCRE[3] szerinti.
- Példák:

```
iex> Regex.run ~r{[cdr]}, "madárcsicsergés"
["d"]
iex> Regex.scan ~r{[cdr]}, "madárcsicsergés"
[["d"], ["r"], ["c"], ["c"], ["r"]]
iex> Regex.split ~r{[cdr]}, "madárcsicsergés"
["ma", "á", "", "si", "se", "gés"]
iex>
```

[2] A `~r{...}` jelölés is egy *szigil*, azaz bűvös jelölés. A szigilekről részletek a Kernel dokumentációjában találhatók.

[3] Perl Compatible Regular Expressions, `http://www.pcre.org`

Reguláris kifejezés (Regex) 1

- Az Elixirben a reguláris kifejezés is önálló típus
- Jelölés: `~r{regexp}`[2] vagy `~r{regexp}options`
- A reguláris kifejezés szintaxisa a PCRE[3] szerinti.
- Példák:

```
iex> Regex.run ~r{[cdr]}, "madárcsicsergés"
["d"]
iex> Regex.scan ~r{[cdr]}, "madárcsicsergés"
[["d"], ["r"], ["c"], ["c"], ["r"]]
iex> Regex.split ~r{[cdr]}, "madárcsicsergés"
["ma", "á", "", "si", "se", "gés"]
iex> Regex.replace ~r{[cdr]}, "madárcsicsergés", "."
"ma.á..si.se.gés"
```

- További részletek a *Regex* modul dokumentációjában

[2] A `~r{...}` jelölés is egy *szigil*, azaz bűvös jelölés. A szigilekről részletek a Kernel dokumentációjában találhatók.

[3] Perl Compatible Regular Expressions, `http://www.pcre.org`

Reguláris kifejezés (Regex) 2

- A regexp után egy vagy több egykarakteres opció állhat

Jel	Jelentés
f	Többsoros sztring első sorában kezdődjön az illesztés
i	Az illesztés ne különböztesse meg a kis- és nagybetűket
m	Többsoros sztring esetén a ^ és a $ az egyes sorok elejét és végét jelentse (a \A és \z jelentése változatlanul a sztring eleje és vége)
s	A . illeszkedjen az újsor-karakterekre is
U	Az egyébként mohó * és + módosítók legyenek lusták, azaz a minta a lehető leghosszabb karaktersorozat helyett a lehető legrövidebbre illeszkedjen
u	Engedje meg Unicode-specifikus minták, pl. \p használatát
x	Engedje meg a bővített mód használatát: ignorálja a szóköz-jellegű (ún. whitespace) karaktereket és a kommenteket (a # jeltől a sor végéig)

Reguláris kifejezés (Regex) 2

- A regexp után egy vagy több egykarakteres opció állhat

Jel	Jelentés
f	Többsoros sztring első sorában kezdődjön az illesztés
i	Az illesztés ne különböztesse meg a kis- és nagybetűket
m	Többsoros sztring esetén a ^ és a $ az egyes sorok elejét és végét jelentse (a \A és \z jelentése változatlanul a sztring eleje és vége)
s	A . illeszkedjen az újsor-karakterekre is
U	Az egyébként mohó * és + módosítók legyenek lusták, azaz a minta a lehető leghosszabb karaktersorozat helyett a lehető legrövidebbre illeszkedjen
u	Engedje meg Unicode-specifikus minták, pl. \p használatát
x	Engedje meg a bővített mód használatát: ignorálja a szóköz-jellegű (ún. whitespace) karaktereket és a kommenteket (a # jeltől a sor végéig)

- Példák:

```
iex> Regex.run ~r{cs.*s}, "Madarak Csicsergése"
["csergés"]
iex> Regex.run ~r{cs.*s}i, "Madarak Csicsergése"
["Csicsergés"]
iex> Regex.run ~r{cs.*s}iU, "Madarak Csicsergése"
["Csics"]
```

Tartalom

Term

- A *term* tetszőleges adatstruktúra
- Minden termnek van *értéke* és *típusa*
- A term maga is kifejezés
- Közelítő rekurzív definíciója:
  Szám-, atom-, függvény- és más értékekből, ill. *termekből* konstruktorokkal felépített, *tovább nem egyszerűsíthető* kifejezés

Term

- A *term* tetszőleges adatstruktúra
- Minden termnek van *értéke* és *típusa*
- A term maga is kifejezés
- Közelítő rekurzív definíciója:
  Szám-, atom-, függvény- és más értékekből, ill. *termekből* konstruktorokkal felépített, *tovább nem egyszerűsíthető* kifejezés
- Példák
  - `kötött = 2021`
  - Term: tovább nem egyszerűsíthető, *tömör*, ha kiértékelhető, azaz nincs benne szabad változó

    ```
    123456789
    {'Diák Detti', [{:khf, [:prolog, :elixir, :prolog]}]}
    [&:erlang.+/2, kötött, fn(x,y) -> x*y end]
    ```

Term

- A *term* tetszőleges adatstruktúra
- Minden termnek van *értéke* és *típusa*
- A term maga is kifejezés
- Közelítő rekurzív definíciója:
  Szám-, atom-, függvény- és más értékekből, ill. *termekből* konstruktorokkal felépített, *tovább nem egyszerűsíthető* kifejezés
- Példák
  - `kötött = 2021`
  - Term: tovább nem egyszerűsíthető, *tömör*, ha kiértékelhető, azaz nincs benne szabad változó

    ```
    123456789
    {'Diák Detti', [{:khf, [:prolog, :elixir, :prolog]}]}
    [&:erlang.+/2, kötött, fn(x,y) -> x*y end]
    ```
  - Nem term (tovább egyszerűsíthető vagy nem tömör)

    ```
    5+6                    # műveletet tartalmaz
    (&:erlang.+/2).(5,6)   # függvényalkalmazást tartalmaz
    szabad                 # szabad változó
    ```

Azonosító (identifier)

- Kisbetűvel vagy aláhúzásjellel (_) kezdődő, betűket, számjegyeket[4] és aláhúzásjeleket tartalmazó, opcionálisan kérdő- vagy felkiáltójellel végződő karaktersorozat
- Konvenció szerint a ?-lel végződő azonosító kiértékelése igazságértéket ad eredményül, a !-lel végződő kiértékelése pedig kivételt dob, ha meghiúsul
- Konvenció szerint az azonosító részeit aláhúzásjellel tagoljuk (ún. megengedő snake_case), vö. atom szintaxisa

[4] UTF-8 kódolású betű, ill. decimális számjegy; lásd `https://hexdocs.pm/elixir/unicode-syntax.html`

Azonosító (identifier)

- Kisbetűvel vagy aláhúzásjellel (_) kezdődő, betűket, számjegyeket[4] és aláhúzásjeleket tartalmazó, opcionálisan kérdő- vagy felkiáltójellel végződő karaktersorozat
- Konvenció szerint a ?-lel végződő azonosító kiértékelése igazságértéket ad eredményül, a !-lel végződő kiértékelése pedig kivételt dob, ha meghiúsul
- Konvenció szerint az azonosító részeit aláhúzásjellel tagoljuk (ún. megengedő snake_case), vö. atom szintaxisa
- Példák:

```
what_s_in_a_name   name?   exec!
_unused   rómeó_és_Júlia   year_2021
```

- Az azonosító változót vagy függvénynevet jelöl

[4] UTF-8 kódolású betű, ill. decimális számjegy; lásd `https://hexdocs.pm/elixir/unicode-syntax.html`

Változó

- Egy változó lehet *szabad* vagy *kötött*
- A szabad változónak nincs értéke, típusa
- A kötött változó valamely konkrét term *szinonimája*

Változó

- Egy változó lehet *szabad* vagy *kötött*
- A szabad változónak nincs értéke, típusa
- A kötött változó valamely konkrét term *szinonimája*
- A változóhoz köthető új érték, de ez korábbi felhasználását nem módosítja
- A ^ (pin) operátor a kötött változó értékét fixálja: nem köthető új értékhez

Változó

- Egy változó lehet *szabad* vagy *kötött*
- A szabad változónak nincs értéke, típusa
- A kötött változó valamely konkrét term *szinonimája*
- A változóhoz köthető új érték, de ez korábbi felhasználását nem módosítja
- A ^ (pin) operátor a kötött változó értékét fixálja: nem köthető új értékhez
- Példák

```
iex> x = fn(x) -> 2*x end # a külső és a belső x nem ugyanaz!
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> y = x
#Function<44.40011524/1 in :erl_eval.expr/5>
iex>
```

Változó

- Egy változó lehet *szabad* vagy *kötött*
- A szabad változónak nincs értéke, típusa
- A kötött változó valamely konkrét term *szinonimája*
- A változóhoz köthető új érték, de ez korábbi felhasználását nem módosítja
- A ^ (pin) operátor a kötött változó értékét fixálja: nem köthető új értékhez
- Példák

```
iex> x = fn(x) -> 2*x end # a külső és a belső x nem ugyanaz!
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> y = x
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> ^x = y.(2)
** (MatchError) no match of right hand side value: 4
iex>
```

Változó

- Egy változó lehet *szabad* vagy *kötött*
- A szabad változónak nincs értéke, típusa
- A kötött változó valamely konkrét term *szinonimája*
- A változóhoz köthető új érték, de ez korábbi felhasználását nem módosítja
- A ^ (pin) operátor a kötött változó értékét fixálja: nem köthető új értékhez
- Példák

```
iex> x = fn(x) -> 2*x end # a külső és a belső x nem ugyanaz!
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> y = x
#Function<44.40011524/1 in :erl_eval.expr/5>
iex> ^x = y.(2)
** (MatchError) no match of right hand side value: 4
iex> x = y.(2)
4
iex> y
#Function<44.40011524/1 in :erl_eval.expr/5>
```

Változó hatásköre, komment, igazságérték

- Változó hatásköre: lexikális
  - A függvény törzsében és fejében definiált változók (utóbbiak másnéven: formális paraméterek) lokálisak a függvényre nézve
  - Modulban is lehet változót definiálni, ami csak modulszinten látható, a modulban definiált függvényekből nem
  - `with` kifejezéssel is definiálhatunk lokális változót, például

```
iex> with a = 5, b = 7, do: a*a + 2*a*b + b*b
144
iex>
```

Változó hatásköre, komment, igazságérték

- Változó hatásköre: lexikális
  - A függvény törzsében és fejében definiált változók (utóbbiak másnéven: formális paraméterek) lokálisak a függvényre nézve
  - Modulban is lehet változót definiálni, ami csak modulszinten látható, a modulban definiált függvényekből nem
  - `with` kifejezéssel is definiálhatunk lokális változót, például

```
iex> with a = 5, b = 7, do: a*a + 2*a*b + b*b
144
iex> a = 11; with a = 5, b = 7, do: a*a + 2*a*b + b*b; a
11
```

Változó hatásköre, komment, igazságérték

- Változó hatásköre: lexikális
  - A függvény törzsében és fejében definiált változók (utóbbiak másnéven: formális paraméterek) lokálisak a függvényre nézve
  - Modulban is lehet változót definiálni, ami csak modulszinten látható, a modulban definiált függvényekből nem
  - `with` kifejezéssel is definiálhatunk lokális változót, például

```
iex> with a = 5, b = 7, do: a*a + 2*a*b + b*b
144
iex> a = 11; with a = 5, b = 7, do: a*a + 2*a*b + b*b; a
11
```

- Komment: # jellel kezdődik, a sor végéig tart

Változó hatásköre, komment, igazságérték

- Változó hatásköre: lexikális
  - A függvény törzsében és fejében definiált változók (utóbbiak másnéven: formális paraméterek) lokálisak a függvényre nézve
  - Modulban is lehet változót definiálni, ami csak modulszinten látható, a modulban definiált függvényekből nem
  - `with` kifejezéssel is definiálhatunk lokális változót, például

```
iex> with a = 5, b = 7, do: a*a + 2*a*b + b*b
144
iex> a = 11; with a = 5, b = 7, do: a*a + 2*a*b + b*b; a
11
```

- Komment: # jellel kezdődik, a sor végéig tart
- Igazságérték, másnéven logikai érték (boolean)
  - Három atomot tekintünk igazságértéknek: `:true`, `:false`, `:nil`
  - Mindhárom írható kettőspont nélkül is: `true`, `false`, `nil`
  - A `false` és `nil` *hamis*, minden más érték (nemcsak a `true`) *igaz*

Változó hatásköre, komment, igazságérték

- Változó hatásköre: lexikális
  - A függvény törzsében és fejében definiált változók (utóbbiak másnéven: formális paraméterek) lokálisak a függvényre nézve
  - Modulban is lehet változót definiálni, ami csak modulszinten látható, a modulban definiált függvényekből nem
  - `with` kifejezéssel is definiálhatunk lokális változót, például

```
iex> with a = 5, b = 7, do: a*a + 2*a*b + b*b
144
iex> a = 11; with a = 5, b = 7, do: a*a + 2*a*b + b*b; a
11
```

- Komment: # jellel kezdődik, a sor végéig tart
- Igazságérték, másnéven logikai érték (boolean)
  - Három atomot tekintünk igazságértéknek: `:true`, `:false`, `:nil`
  - Mindhárom írható kettőspont nélkül is: `true`, `false`, `nil`
  - A `false` és `nil` *hamis*, minden más érték (nemcsak a `true`) *igaz*
  - Angolul szokás megkülönböztetni a *true*-t a *truthy*-tól, a *false*-t a *falsy*-tól, pl. JavaScript, Java, Elixir.

II. rész

Műveletek listákon

Tartalom

Műveletek listákon 1

- A lista láncolt lineáris adatstruktúra, ezért olcsó az első elemét (a *fejét*) és az összes többi elemét (a *farkát*) megkapni, de drága az utolsó elemét elérni, mert végig kell gyalogolni a listán

Műveletek listákon 1

- A lista láncolt lineáris adatstruktúra, ezért olcsó az első elemét (a *fejét*) és az összes többi elemét (a *farkát*) megkapni, de drága az utolsó elemét elérni, mert végig kell gyalogolni a listán
- A funkcionális nyelvekben, a többi adatstruktúrával egyezően, a lista nem frissíthető, az Elixirben sem: amikor a lista egy elemét le akarjuk cserélni, akkor *másolatot* kell készítenünk a lecserélendő elem előtti részlistáról
- A másolás során a lecserélendő elem előtti összes elemet félre kell raknunk, majd a lecserélendő elem utáni *megosztott* farokrész elé be kell fűznünk az új elemet, ezt követően pedig a félrerakott elemeket egyesével be kell fűznünk az új elemet már tartalmazó listarész elé
- Vagyis a lista adott elemi utáni farkáról nem készül másolat, mert az Elixir *megosztja* a lista farkát a régi és az új elemet tartalmazó listák között

Műveletek listákon 1

- A lista láncolt lineáris adatstruktúra, ezért olcsó az első elemét (a *fejét*) és az összes többi elemét (a *farkát*) megkapni, de drága az utolsó elemét elérni, mert végig kell gyalogolni a listán
- A funkcionális nyelvekben, a többi adatstruktúrával egyezően, a lista nem frissíthető, az Elixirben sem: amikor a lista egy elemét le akarjuk cserélni, akkor *másolatot* kell készítenünk a lecserélendő elem előtti részlistáról
- A másolás során a lecserélendő elem előtti összes elemet félre kell raknunk, majd a lecserélendő elem utáni *megosztott* farokrész elé be kell fűznünk az új elemet, ezt követően pedig a félrerakott elemeket egyesével be kell fűznünk az új elemet már tartalmazó listarész elé
- Vagyis a lista adott elemi utáni farkáról nem készül másolat, mert az Elixir *megosztja* a lista farkát a régi és az új elemet tartalmazó listák között
- A lista annyira megkerülhetetlen adatstruktúra a funkcionális nyelvekben, hogy már eddig is sok példát láttunk a használatára. A következő dián összefoglaljuk a leggyakoribb listaműveleteket
- A sztring ugyan nem listaként van ábrázolva az Elixirben, de a használata hasonló, ezért a leggyakoribb sztringműveleteket is összefoglaljuk később egy dián

Műveletek listákon 2

- Lista feje, farka, hossza: `hd(xs)`, `tl(xs)`, `length(xs)`[5]
- Két lista összefűzése (konkatenációja): `xs ++ ys`, eredménye `xs` összes eleme `ys` elé fűzve az eredeti sorrendben
- Két lista különbsége: `xs -- ys`, eredménye `xs` azon elemeinek listája az eredeti sorrendben, amelyek nincsenek benne `ys`-ben
- Tagsági vizsgálat: `x in xs` eredménye `true`, ha `x` eleme `xs`-nek

[5] `hd/1`, `tl/1`, `length/1`, `++/2`, `--/2`, `in/2` a Kernel modulban vannak definiálva

Műveletek listákon 2

- Lista feje, farka, hossza: `hd(xs)`, `tl(xs)`, `length(xs)`[5]
- Két lista összefűzése (konkatenációja): `xs ++ ys`, eredménye `xs` összes eleme `ys` elé fűzve az eredeti sorrendben
- Két lista különbsége: `xs -- ys`, eredménye `xs` azon elemeinek listája az eredeti sorrendben, amelyek nincsenek benne `ys`-ben
- Tagsági vizsgálat: `x in xs` eredménye `true`, ha `x` eleme `xs`-nek
- Példák

```
iex> [:a, 'a', [65]] ++ [1+2, 2/1, 'a'] # 65 == ?A
[:a, 'a', 'A', 3, 2.0, 'a']
iex>
```

[5] `hd/1`, `tl/1`, `length/1`, `++/2`, `--/2`, `in/2` a Kernel modulban vannak definiálva

Műveletek listákon 2

- Lista feje, farka, hossza: `hd(xs)`, `tl(xs)`, `length(xs)`[5]
- Két lista összefűzése (konkatenációja): `xs ++ ys`, eredménye `xs` összes eleme `ys` elé fűzve az eredeti sorrendben
- Két lista különbsége: `xs -- ys`, eredménye `xs` azon elemeinek listája az eredeti sorrendben, amelyek nincsenek benne `ys`-ben
- Tagsági vizsgálat: `x in xs` eredménye `true`, ha `x` eleme `xs`-nek
- Példák

```
iex> [:a, 'a', [65]] ++ [1+2, 2/1, 'a'] # 65 == ?A
[:a, 'a', 'A', 3, 2.0, 'a']
iex> Enum.to_list(1..100000) ++ [100001] # rossz hatékonyságú!
[1, 2, 3, 4, 5, 6, 7, 8, ...100001]
iex>
```

[5] `hd/1`, `tl/1`, `length/1`, `++/2`, `--/2`, `in/2` a Kernel modulban vannak definiálva

Műveletek listákon 2

- Lista feje, farka, hossza: `hd(xs)`, `tl(xs)`, `length(xs)`[5]
- Két lista összefűzése (konkatenációja): `xs ++ ys`, eredménye `xs` összes eleme `ys` elé fűzve az eredeti sorrendben
- Két lista különbsége: `xs -- ys`, eredménye `xs` azon elemeinek listája az eredeti sorrendben, amelyek nincsenek benne `ys`-ben
- Tagsági vizsgálat: `x in xs` eredménye `true`, ha `x` eleme `xs`-nek
- Példák

```
iex> [:a, 'a', [65]] ++ [1+2, 2/1, 'a'] # 65 == ?A
[:a, 'a', 'A', 3, 2.0, 'a']
iex> Enum.to_list(1..100000) ++ [100001] # rossz hatékonyságú!
[1, 2, 3, 4, 5, 6, 7, 8, ...100001]
iex> [:a, 'a', [65], 'a'] -- ["A", 2/1, 'a']
[:a, 'A', 'a']
iex> [:a, 'a', [65], 'a'] -- ["A", 2/1, 'a', :a, :a, :a]
['A', 'a']
iex> [1, 2, 3] -- [1.0, 2] # szigorú egyenlőség: 1 ≠ 1.0
[1, 3]
iex>
```

[5] `hd/1`, `tl/1`, `length/1`, `++/2`, `--/2`, `in/2` a Kernel modulban vannak definiálva

Műveletek listákon 2

- Lista feje, farka, hossza: `hd(xs)`, `tl(xs)`, `length(xs)`[5]
- Két lista összefűzése (konkatenációja): `xs ++ ys`, eredménye `xs` összes eleme `ys` elé fűzve az eredeti sorrendben
- Két lista különbsége: `xs -- ys`, eredménye `xs` azon elemeinek listája az eredeti sorrendben, amelyek nincsenek benne `ys`-ben
- Tagsági vizsgálat: `x in xs` eredménye `true`, ha `x` eleme `xs`-nek
- Példák

```
iex> [:a, 'a', [65]] ++ [1+2, 2/1, 'a'] # 65 == ?A
[:a, 'a', 'A', 3, 2.0, 'a']
iex> Enum.to_list(1..100000) ++ [100001] # rossz hatékonyságú!
[1, 2, 3, 4, 5, 6, 7, 8, ...100001]
iex> [:a, 'a', [65], 'a'] -- ["A", 2/1, 'a']
[:a, 'A', 'a']
iex> [:a, 'a', [65], 'a'] -- ["A", 2/1, 'a', :a, :a, :a]
['A', 'a']
iex> [1, 2, 3] -- [1.0, 2] # szigorú egyenlőség: 1 ≠ 1.0
[1, 3]
iex> "A" in ["A", 2/1, 'a', :a, :a, :a]
true
```

[5] `hd/1`, `tl/1`, `length/1`, `++/2`, `--/2`, `in/2` a Kernel modulban vannak definiálva

Műveletek listákon 3

- Lista első / utolsó eleme; ha nincs, `default` vagy `nil`:
  `first(list, default \\ nil)`, `last(list, default \\ nil)`
- Egy elem első előfordulásának törlésével kapott lista:
  `delete(list, elem)`
- Adott pozíciójú elem törlésével / beszúrásával / cseréjével kapott lista:
  `delete_at(list, index)`, `insert_at(list, index, value)`,
  `replace_at(list, index, value)`, `update_at(list, index, fun)`
  Indexelés 0-tól, negatív index a lista végéről indul. `update_at/3` a `fun` függvényt alkalmazza az adott pozíciójú elemre.
- Lista kilapításával / kilapítása után a `tail` elé fűzésével kapott lista:
  `flatten(list)`, `flatten(list, tail)`
- Elem többszörözésével kapott lista: `duplicate(elem, n)`

Műveletek listákon 3

- Lista első / utolsó eleme; ha nincs, `default` vagy `nil`:
  `first(list, default \\ nil)`, `last(list, default \\ nil)`
- Egy elem első előfordulásának törlésével kapott lista:
  `delete(list, elem)`
- Adott pozíciójú elem törlésével / beszúrásával / cseréjével kapott lista:
  `delete_at(list, index)`, `insert_at(list, index, value)`,
  `replace_at(list, index, value)`, `update_at(list, index, fun)`
  Indexelés 0-tól, negatív index a lista végéről indul. `update_at/3` a `fun` függvényt alkalmazza az adott pozíciójú elemre.
- Lista kilapításával / kilapítása után a `tail` elé fűzésével kapott lista:
  `flatten(list)`, `flatten(list, tail)`
- Elem többszörözésével kapott lista: `duplicate(elem, n)`
- Listák listájából ennesek listája: `zip(list_of_lists)`
- Példák:

```
iex> List.zip([[:a, :b, :c], [:d, :e, :f, :g, :h], [:i, :j, :k, :l]])
[{:a, :d, :i}, {:b, :e, :j}, {:c, :f, :k}] # Listák vége levágva
iex>
```

Műveletek listákon 3

- Lista első / utolsó eleme; ha nincs, `default` vagy `nil`:
  `first(list, default \\ nil)`, `last(list, default \\ nil)`
- Egy elem első előfordulásának törlésével kapott lista:
  `delete(list, elem)`
- Adott pozíciójú elem törlésével / beszúrásával / cseréjével kapott lista:
  `delete_at(list, index)`, `insert_at(list, index, value)`,
  `replace_at(list, index, value)`, `update_at(list, index, fun)`
  Indexelés 0-tól, negatív index a lista végéről indul. `update_at/3` a `fun` függvényt alkalmazza az adott pozíciójú elemre.
- Lista kilapításával / kilapítása után a `tail` elé fűzésével kapott lista:
  `flatten(list)`, `flatten(list, tail)`
- Elem többszörözésével kapott lista: `duplicate(elem, n)`
- Listák listájából ennesek listája: `zip(list_of_lists)`
- Példák:

```
iex> List.zip([[:a, :b, :c], [:d, :e, :f, :g, :h], [:i, :j, :k, :l]])
[{:a, :d, :i}, {:b, :e, :j}, {:c, :f, :k}] # Listák vége levágva
iex> List.flatten(['abc', [['defgh']], ['ijkl']], 'zzz')
  'abcdefghijklzzz'
```

Műveletek listákon 4

- Konverziós függvények, pl. `List.to_string`, `Tuple.to_list`
- Van három tesztelő függvény is:
  - `improper?(list)` igaz, ha `list` *nem valódi* lista, azaz egy listakonstruktorban a farok *nem* lista, pl. `[1,2|3]`, `[:a,:b|nil]`
  - `starts_with?(list, prefix)` igaz, ha `list` `prefix`-szel kezdődik
  - `ascii_printable?(list, n \\ :infinity)` igaz, ha `list` első `n` karaktere 7-bites ASCII-kódolású és nyomtatható, beleértve a vezérlő karaktereket is (`\a`, `\b`, `\t`, `\n`, `\v`, `\f`, `\r`, `\e`)

Műveletek listákon 4

- Konverziós függvények, pl. `List.to_string`, `Tuple.to_list`
- Van három tesztelő függvény is:
  - `improper?(list)` igaz, ha `list` *nem valódi* lista, azaz egy listakonstruktorban a farok *nem* lista, pl. `[1,2|3]`, `[:a,:b|nil]`
  - `starts_with?(list, prefix)` igaz, ha `list` `prefix`-szel kezdődik
  - `ascii_printable?(list, n \\ :infinity)` igaz, ha `list` első `n` karaktere 7-bites ASCII-kódolású és nyomtatható, beleértve a vezérlő karaktereket is (`\a`, `\b`, `\t`, `\n`, `\v`, `\f`, `\r`, `\e`)

Az `Enum` modul függvényei is alkalmazhatók listákra:

- Lista megfordításával / megfordítása után a `tail` elé fűzésével kapott lista: `reverse(list)`, `reverse(list, tail)`
- Lista adott indexű eleme, ha nincs ilyen, `default` vagy `nil`: `at(list, index, default \\ nil)`

Műveletek listákon 4

- Konverziós függvények, pl. `List.to_string`, `Tuple.to_list`
- Van három tesztelő függvény is:
  - `improper?(list)` igaz, ha `list` *nem valódi* lista, azaz egy listakonstruktorban a farok *nem* lista, pl. `[1,2|3]`, `[:a,:b|nil]`
  - `starts_with?(list, prefix)` igaz, ha `list` `prefix`-szel kezdődik
  - `ascii_printable?(list, n \\ :infinity)` igaz, ha `list` első `n` karaktere 7-bites ASCII-kódolású és nyomtatható, beleértve a vezérlő karaktereket is (`\a`, `\b`, `\t`, `\n`, `\v`, `\f`, `\r`, `\e`)

Az `Enum` modul függvényei is alkalmazhatók listákra:

- Lista megfordításával / megfordítása után a `tail` elé fűzésével kapott lista: `reverse(list)`, `reverse(list, tail)`
- Lista adott indexű eleme, ha nincs ilyen, `default` vagy `nil`: `at(list, index, default \\ nil)`

```
iex> List.starts_with? 'almafa', [?a, ?l]
true
```

```
iex> Enum.reverse 'almafa'
'afamla'
iex> Enum.at 'almafa', 2
?m
```

Műveletek listákon 5

További függvények az `Enum` modulból:

- Lista legkisebb / legnagyobb eleme:
  ```
  min(list, sorter \\ &<=/2,
            empty_fallback \\ fn -> raise(Enum.EmptyError) end)
  ```
  `max/3` paraméterezése hasonló, `&<=/2` helyett `&>=/2`-vel.
  Ha `list` üres, a 3. paraméterként átadott *függvény* aktivizálódik.
- Lista `n` elemű eleje, `n` elem utáni farka: `take(list, n)`, `drop(list, n)`
  Ha `n` negatív, az elemeket a lista végéről kezdve emeli le / dobja el.
- Lista részlistája: `slice(list, range)` a `range` tartományba eső indexű elemek listája / `slice(list, start, n)` a `start` indextől kezdődő `n` elemű részlista. Ha `range`, ill. `start` negatív, indexelés a lista végéről.

Műveletek listákon 5

További függvények az `Enum` modulból:

- Lista legkisebb / legnagyobb eleme:
  `min(list, sorter \\ &<=/2, empty_fallback \\ fn -> raise(Enum.EmptyError) end)`
  `max/3` paraméterezése hasonló, `&<=/2` helyett `&>=/2`-vel.
  Ha `list` üres, a 3. paraméterként átadott *függvény* aktivizálódik.
- Lista `n` elemű eleje, `n` elem utáni farka: `take(list, n)`, `drop(list, n)`
  Ha `n` negatív, az elemeket a lista végéről kezdve emeli le / dobja el.
- Lista részlistája: `slice(list, range)` a `range` tartományba eső indexű elemek listája / `slice(list, start, n)` a `start` indextől kezdődő `n` elemű részlista. Ha `range`, ill. `start` negatív, indexelés a lista végéről.

Példák

```
iex> {(Enum.max 'mióta') === ?ó, (Enum.min [], fn -> 0 end)}
{true, 0}
iex> xs='indulakutyasatyukaludni'; {(Enum.take xs,-5), (Enum.drop xs,5)}
{'ludni', 'akutyasatyukaludni'}
iex> {(Enum.slice xs, 6..10), (Enum.slice xs, -10..-7)}
{'kutya', 'tyuk'}
```

Műveletek listákon 6

És még néhány függvény az `Enum` modulból:

- Lista kettévágva: `split(list, n)` ugyanaz, csak rövidebben mint `{(take list, n), (drop list, n)}`
- Lista rendezve alapértelmezés / `fun` függény szerint: `sort(list)`, `sort(list, fun)`
- Lista többszörös értékek nélkül: `uniq(list)`

Műveletek listákon 6

És még néhány függvény az `Enum` modulból:

- Lista kettévágva: `split(list, n)` ugyanaz, csak rövidebben mint `{(take list, n), (drop list, n)}`
- Lista rendezve alapértelmezés / `fun` függény szerint: `sort(list)`, `sort(list, fun)`
- Lista többszörös értékek nélkül: `uniq(list)`

Példák

```
iex> xs='indulakutyasatyukaludni'; [(Enum.split xs,5),(Enum.split xs,-5)]
[{'indul', 'akutyasatyukaludni'}, {'indulakutyasatyuka', 'ludni'}]
iex> Enum.sort xs
'aaaaddiikkllnnsttuuuuyy'
iex> Enum.sort xs, &>=/2
'yyuuuuttsnnllkkiiddaaaa'
iex> Enum.uniq xs
'indulaktys'
```

Műveletek listákon 6

És még néhány függvény az `Enum` modulból:

- Lista kettévágva: `split(list, n)` ugyanaz, csak rövidebben mint `{(take list, n), (drop list, n)}`
- Lista rendezve alapértelmezés / `fun` függény szerint: `sort(list)`, `sort(list, fun)`
- Lista többszörös értékek nélkül: `uniq(list)`

Példák

```
iex> xs='indulakutyasatyukaludni'; [(Enum.split xs,5),(Enum.split xs,-5)]
[{'indul', 'akutyasatyukaludni'}, {'indulakutyasatyuka', 'ludni'}]
iex> Enum.sort xs
'aaaaddiikkllnnsttuuuuyy'
iex> Enum.sort xs, &>=/2
'yyuuuuttsnnllkkiiddaaaa'
iex> Enum.uniq xs
'indulaktys'
```

Az `Enum` modul függvényei – mind *mohó kiértékelésű* – egyéb korlátos, felsorolható (enumerable) adatstruktúrákra is alkalmazhatók.
Nem korlátos adatstruktúrákra a `Stream` modul függvényeit – ezek *lusta kiértékelésűek* – lehet használni

Tartalom

Listakezelés – rövid példák 1

```
iex> xs = [10,20,30] # mintaillesztés és változó kötése értékhez
[10, 20, 30]
iex> x = hd xs       # hd: lista feje
10
iex> rs = tl xs      # tl: lista farka
[20, 30]
iex> {zs,xs} = {xs,[5,6]} # mintaillesztés és változó újrakötése
{[10, 20, 30], [5, 6]}
iex> xs              # xs-hez új értéket kötöttünk
[5, 6]
iex> zs              # xs változott, zs nem!
[10, 20, 30]
iex> ^xs = [7,8,9]   # ^: változó 'fixálása', csak mintaillesztés, kötés nélkül
** (MatchError) no match of right hand side value: ~c"\a\b\t"
iex> hd tl xs        # összetett kifejezés is kiértékelhető
6
iex> tl []           # mi az üres lista farka?
** (ArgumentError) errors were found at the given arguments:
  * 1st argument: not a nonempty list
    :erlang.tl([])
```

Listakezelés – rövid példák 2

```
iex> for i <- 7..13, do: [i]
[~c"\a", ~c"\b", ~c"\t", ~c"\n", ~c"\v", ~c"\f", ~c"\r"]
iex> for i <- 27..27, do: [i]
[~c"\e"]
iex> for i <- 32..126, do: [i]
[~c" ", ~c"!", ~c"\"", ~c"#", ~c"$", ~c"%", ~c"&", ~c"'", ~c"(", ~c")", ~c"*",
 ~c"+", ~c",", ~c"-", ~c".", ~c"/", ~c"0", ~c"1", ~c"2", ~c"3", ~c"4", ~c"5",
 ~c"6", ~c"7", ~c"8", ~c"9", ~c":", ~c";", ~c"<", ~c"=", ~c">", ~c"?", ~c"@",
 ~c"A", ~c"B", ~c"C", ~c"D", ~c"E", ~c"F", ~c"G", ~c"H", ~c"I", ~c"J", ~c"K",
 ~c"L", ~c"M", ~c"N", ~c"O", ~c"P", ~c"Q", ...]
iex> IO.puts (for i <- 35..126, do: [i])
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
:ok
iex> IO.inspect (for i <- 32..126, do: [i]), limit: :infinity
[~c" ", ~c"!", ~c"\"", ~c"#", ~c"$", ~c"%", ~c"&", ~c"'", ~c"(", ~c")", ~c"*",
 ~c"+", ~c",", ~c"-", ~c".", ~c"/", ~c"0", ~c"1", ~c"2", ~c"3", ~c"4", ~c"5",
 ~c"6", ~c"7", ~c"8", ~c"9", ~c":", ~c";", ~c"<", ~c"=", ~c">", ~c"?", ~c"@",
 ~c"A", ~c"B", ~c"C", ~c"D", ~c"E", ~c"F", ~c"G", ~c"H", ~c"I", ~c"J", ~c"K",
 ~c"L", ~c"M", ~c"N", ~c"O", ~c"P", ~c"Q", ~c"R", ~c"S", ~c"T", ~c"U", ~c"V",
 ~c"W", ~c"X", ~c"Y", ~c"Z", ~c"[", ~c"\", ~c"]", ~c"^", ~c"_", ~c"`", ~c"a",
 ~c"b", ~c"c", ~c"d", ~c"e", ~c"f", ~c"g", ~c"h", ~c"i", ~c"j", ~c"k", ~c"l",
 ~c"m", ~c"n", ~c"o", ~c"p", ~c"q", ~c"r", ~c"s", ~c"t", ~c"u", ~c"v", ~c"w",
 ~c"x", ~c"y", ~c"z", ~c"{", ~c"|", ~c"}", ~c"~"]
 [~c" ", ~c"!", ~c"\"", ~c"#", ~c"$", ~c"%", ~c"&", ~c"'", ~c"(", ~c")", ~c"*",
  ~c"+", ~c",", ~c"-", ~c".", ~c"/", ~c"0", ~c"1", ~c"2", ~c"3", ~c"4", ~c"5",
  ~c"6", ~c"7", ~c"8", ~c"9", ~c":", ~c";", ~c"<", ~c"=", ~c">", ~c"?", ~c"@",
  ~c"A", ~c"B", ~c"C", ~c"D", ~c"E", ~c"F", ~c"G", ~c"H", ~c"I", ~c"J", ~c"K",
  ~c"L", ~c"M", ~c"N", ~c"O", ~c"P", ~c"Q", ...]
```

Listakezelés – rövid példák 3

`fpea.ex` – Számlista összege

```
@spec sum(xs::[integer]) :: s::integer
# Az xs számlista összege s
def sum([]), do: 0 # a ", do:" jelölés többsoros változata a "do ... end"
def sum(xs)  do x = hd xs; rs = tl xs; x + sum rs end # újsor helyett ;
```

Listakezelés – rövid példák 3

`fpea.ex` – Számlista összege

```
@spec sum(xs::[integer]) :: s::integer
# Az xs számlista összege s
def sum([]), do: 0 # a ", do:" jelölés többsoros változata a "do ... end"
def sum(xs)  do x = hd xs; rs = tl xs; x + sum rs end # újsor helyett ;
```

```
iex> c "fpea.ex"
[Fpea]
iex> xs = [10, 20.5, 30.5]
[10, 20.5, 30.5]
iex> Fpea.sum xs
61.0
iex> Fpea.sum tl xs
51.0
iex> Fpea.sum(tl(tl(tl xs)))
0
iex>
```

Listakezelés – rövid példák 3

`fpea.ex` – Számlista összege

```
@spec sum(xs::[integer]) :: s::integer
# Az xs számlista összege s
def sum([]), do: 0 # a ", do:" jelölés többsoros változata a "do ... end"
def sum(xs)  do x = hd xs; rs = tl xs; x + sum rs end # újsor helyett ;
```

```
iex> c "fpea.ex"
[Fpea]
iex> xs = [10, 20.5, 30.5]
[10, 20.5, 30.5]
iex> Fpea.sum xs
61.0
iex> Fpea.sum tl xs
51.0
iex> Fpea.sum(tl(tl(tl xs)))
0
iex> Fpea.sum "abc" # "abc" !== [97, 98, 99]: "abc" sztring, nem lista
** (ArgumentError) errors were found at the given arguments:
  * 1st argument: not a nonempty list
    :erlang.hd("abc")
    fpea.ex:13: Fpea.sum/1
iex>
```

Listakezelés – rövid példák 3

fpea.ex – Számlista összege

```
@spec sum(xs::[integer]) :: s::integer
# Az xs számlista összege s
def sum([]), do: 0 # a ", do:" jelölés többsoros változata a "do ... end"
def sum(xs)  do x = hd xs; rs = tl xs; x + sum rs end # újsor helyett ;
```

```
iex> c "fpea.ex"
[Fpea]
iex> xs = [10, 20.5, 30.5]
[10, 20.5, 30.5]
iex> Fpea.sum xs
61.0
iex> Fpea.sum tl xs
51.0
iex> Fpea.sum(tl(tl(tl xs)))
0
iex> Fpea.sum "abc" # "abc" !== [97, 98, 99]: "abc" sztring, nem lista
** (ArgumentError) errors were found at the given arguments:
  * 1st argument: not a nonempty list
    :erlang.hd("abc")
    fpea.ex:13: Fpea.sum/1
iex> Fpea.sum 'abc' # 'abc' === [97, 98, 99]: 'abc' karakterkódok listája
294
```

Listakezelés – rövid példák 4

`fpea.ex` – Két lista összefűzése

```
@spec append(xs::[any], ys::[any]) :: rs::[any]
# rs az xs lista ys elé fűzésével kapott lista
def append([], ys), do: ys
def append(xs, ys), do: [(hd xs) | (append (tl xs), ys)]
@spec revapp(xs::[any], ys::[any]) :: rs::[any]
# rs a megfordított xs lista ys elé fűzésével kapott lista
def revapp([], ys), do: ys
def revapp(xs, ys), do: revapp (tl xs), [(hd xs) | ys]
```

Listakezelés – rövid példák 4

`fpea.ex` – Két lista összefűzése

```
@spec append(xs::[any], ys::[any]) :: rs::[any]
# rs az xs lista ys elé fűzésével kapott lista
def append([], ys), do: ys
def append(xs, ys), do: [(hd xs) | (append (tl xs), ys)]
@spec revapp(xs::[any], ys::[any]) :: rs::[any]
# rs a megfordított xs lista ys elé fűzésével kapott lista
def revapp([], ys), do: ys
def revapp(xs, ys), do: revapp (tl xs), [(hd xs) | ys]
```

```
iex(22)> c "fpea.ex"
[Fpea]
iex(23)> xs
[10, 20.5, 30.5]
iex(24)> Fpea.append(xs, [:a,:b,:c,:d])
[10, 20.5, 30.5, :a, :b, :c, :d]
iex(25)> Fpea.revapp xs, [:a,:b,:c,:d]
[30.5, 20.5, 10, :a, :b, :c, :d]
```

III. rész

Műveletek sztringeken

Tartalom

Műveletek sztringeken 1

A sztringek nem listák az Elixirben – mégcsak nem is kollekciók –, de mivel kényelmes listaszerűen kezelni őket, a `String` modulban vannak ezt lehetővé tevő függvények.
Két fogalmat kell megkülönböztetnünk: a kódpontot (code point) és a grafémát (grapheme cluster, röviden grapheme).

Műveletek sztringeken 1

A sztringek nem listák az Elixirben – mégcsak nem is kollekciók –, de mivel kényelmes listaszerűen kezelni őket, a `String` modulban vannak ezt lehetővé tevő függvények.
Két fogalmat kell megkülönböztetnünk: a kódpontot (code point) és a grafémát (grapheme cluster, röviden grapheme).

- A **kódpont** egyetlen Unicode karakter, egy vagy több bájt ábrázolja

```
iex> {byte_size("á"), String.length("á")}
{2, 1}
```

Műveletek sztringeken 1

A sztringek nem listák az Elixirben – mégcsak nem is kollekciók –, de mivel kényelmes listaszerűen kezelni őket, a `String` modulban vannak ezt lehetővé tevő függvények.
Két fogalmat kell megkülönböztetnünk: a kódpontot (code point) és a grafémát (grapheme cluster, röviden grapheme).

- A **kódpont** egyetlen Unicode karakter, egy vagy több bájt ábrázolja

```
iex> {byte_size("á"), String.length("á")}
{2, 1}
```

- A **graféma** egy vagy több kódpont, ami egyetlen karakternek *látszik*

```
iex> str = "\u0065\u0302"; {byte_size(str), String.length(str)}
{3, 1}
iex> "u\u0302"# U+0302 Combining Circumflex Accent
"û"
iex> String.codepoints(str)
["e", "^"] # Két egykarakteres sztring van a listában.
iex> String.graphemes(str)
["ê"] # Egyetlen egykarakteres sztring van a listában.
```

Műveletek sztringeken 2

- <> a konkatenálás jele: `"ál"<>"om"`
- `string` első / utolsó grafémája, grafémáinak száma: `first(string)`, `last(string)`, `length(string)`
- Graféma `string` `pos` pozíciójában: `at(string, pos)`
- Tartalmazza-e `string` `patts` legalább egy elemét: `contains?(string, patts)`
- `string` elejéről / végéről / mindkettőről levágja a szóköz-jellegű (whitespace) UTF-8 karaktereket: `trim_leading(string)`, `trim_trailing(string)`, `trim(string)`

Műveletek sztringeken 2

- <> a konkatenálás jele: `"ál"<>"om"`
- `string` első / utolsó grafémája, grafémáinak száma: `first(string)`, `last(string)`, `length(string)`
- Graféma `string` `pos` pozíciójában: `at(string, pos)`
- Tartalmazza-e `string` `patts` legalább egy elemét: `contains?(string, patts)`
- `string` elejéről / végéről / mindkettőről levágja a szóköz-jellegű (whitespace) UTF-8 karaktereket: `trim_leading(string)`, `trim_trailing(string)`, `trim(string)`
- Példák

```
iex> str = " "<>" "<>"kutyafüle"<>" "; String.at(str, 8)
"ü"
iex> String.contains?(str, "ü")
true
iex> String.contains?(str, ["ü","ty","n"])
true
iex> String.contains?(str, ["n"])
false
iex> {String.trim_leading(str), String.trim(str)}
{"kutyafüle ", "kutyafüle"}
```

Műveletek sztringeken 3

- Mint a listánál, csak graféma-elemekkel: `slice(string, range)`, `slice(string, start, n)`, `duplicate(string, n)`, `reverse(string)`, `starts_with?(string, prefix)`
- Sztring két darabra vágva adott pozícióban: `split_at(string, pos)`; több darabra szabdalva UTF-8 whitespace-ek mentén: `split(string)`
  - A vezető és záró UTF-8 whitespace-eket ignorálja

Műveletek sztringeken 3

- Mint a listánál, csak graféma-elemekkel: `slice(string, range)`, `slice(string, start, n)`, `duplicate(string, n)`, `reverse(string)`, `starts_with?(string, prefix)`
- Sztring két darabra vágva adott pozícióban: `split_at(string, pos)`; több darabra szabdalva UTF-8 whitespace-ek mentén: `split(string)`
  - A vezető és záró UTF-8 whitespace-eket ignorálja
- Példák

```
iex> str = "indulakutyasatyukaludni"; String.reverse(str)
"indulakuytasaytukaludni"
iex> String.starts_with?(str, "indula")
true
iex> {String.slice(str, 6..10), String.slice(str, -10..-7)}
{"kutya", "tyuk"}
iex> String.duplicate("indul ", 3)
"indul indul indul "
iex> String.split_at(" "<>" "<>"kutya füle macska farka"<>" ", 12)
{"  kutya füle", " macska farka "} # párt ad eredményül
iex> String.split(" "<>" "<>"kutya füle macska farka"<>" ")
["kutya", "füle", "macska", "farka"] # listát ad eredményül
```

IV. rész

Problémamegoldási technikák

Tartalom

Dinamikus programozás

Optimalizálási feladatok megoldására használható módszer. Richard Bellman fejlesztette ki 1950 környékén részben a hadsereg megrendelésére. Az elnevezést a jobb eladhatóság miatt adta neki.

- Az eredetihez hasonló részfeladatokat tűzünk ki, ami lehet akár általánosabb is az eredeti feladatnál.
- A részfeladatokat általában kisebb inputra oldjuk először meg.
- A részfeladatok eredményét eltároljuk.
- Sorba rendezem a feladatokat úgy, hogy a későbbi feladatok megoldásánál fel tudjam használni a korábbiak eredményét.
- Az első néhány részfeladat legyen könnyen megoldható önmagában is.
- A későbbi feladatok eredményét a korábbiak eredményét felhasználva kapjuk. A részfeladatokat úgy határozzuk meg, hogy ez könnyen menjen.
- Az összes részfeladat megoldásából már könnyen megkapható az eredeti feladat megoldása.

Dinamikus programozás: Fibonacci. Benchmarking és debugging

A Fibonacci-számok kiszámítására hatféle algoritmust mutat be a második előadás segédanyaga:

- Elágazó rekurzióval
- Memoizálással (dinamikus programozás felülről lefelé haladva), Elixir Map-pel
- Táblázattal (dinamikus programozás alulról felfelé haladva), Elixir Map-pel
- Táblázattal (dinamikus programozás alulról felfelé haladva), Erlang :array-jel
- Táblázattal (dinamikus programozás alulról felfelé haladva), Elixir List-tel
- Az (n-2)-edik (prev) és az (n-1)-edik (curr) Fibonacci-szám nyilvántartásával

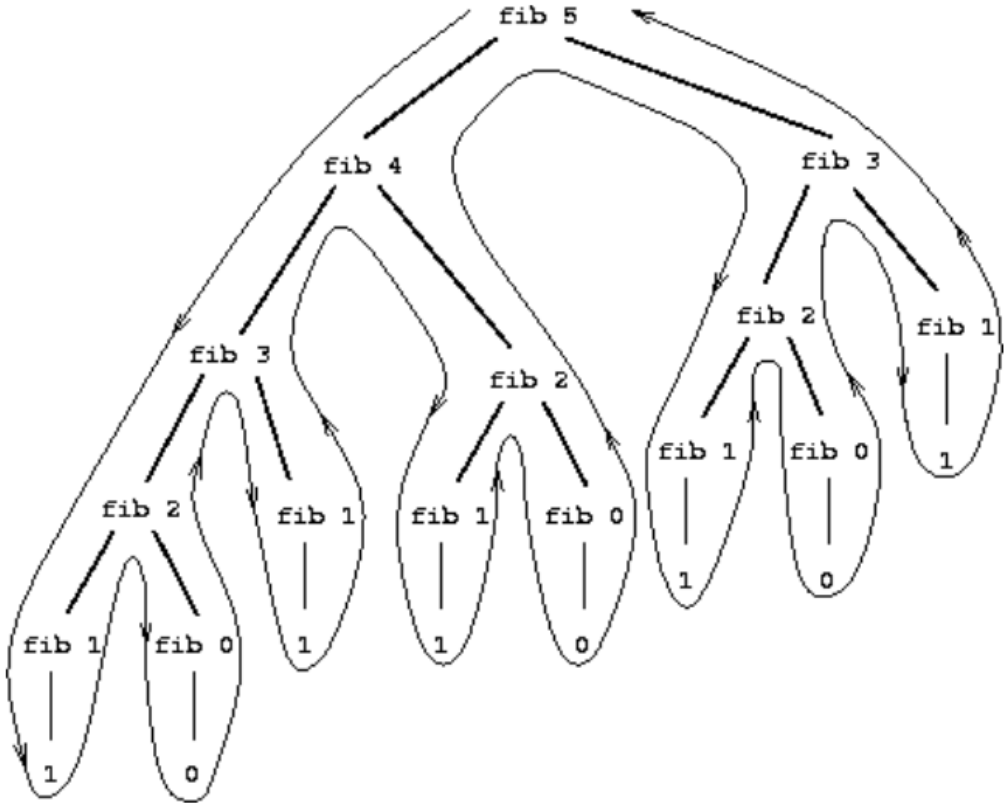
A második előadás segédanyaga a honlapról letölthető:

- `https://dp.iit.bme.hu/dp26a/ea/dp26a-fp2ea-fibonacci.livemd`

A segédanyag a `Benchee` és a `KinoExplorer` modulok használatára is mutat példákat.

Fibonacci elágazó rekurzióval

```
# O(2^n) futási idő, O(2^n) tárhely
@spec fib(i :: integer()) :: n :: integer()
# n az i-edik Fibonacci-szám
def fib(0), do: 0
def fib(1), do: 1
def fib(i), do: fib(i-1) + fib(i-2)
```



2. gyakorlat

- Ezen a héten a dékáni szünet miatt nem találkozunk a gyakorlaton
- **2026. szeptember 22.** 10:15 az IB027 és IE007 termekben
- Alapvetően otthoni munka, a gyakorlaton megbeszéljük a feladatok megoldásait
- Gyakorlat anyaga:
  `https://dp.iit.bme.hu/dp26a/gy/dp26a-fp2gy.livemd`
- Felkészüléshez ajánlott:
  `https://dp.iit.bme.hu/dp26a/ea/dp26a-fp2ea-fibonacci.livemd`