Funkcijsko programiranje

prejšnjič...

Spoznali smo **funkcijske predpise**

$$(x \mapsto e_1)(e_2) = e_1[e_2/x]$$

$$(x \mapsto x^2 + 3 \cdot x + 7)(3) \equiv$$

$$(x^2 + 3 \cdot x + 7)[3/x] \equiv$$

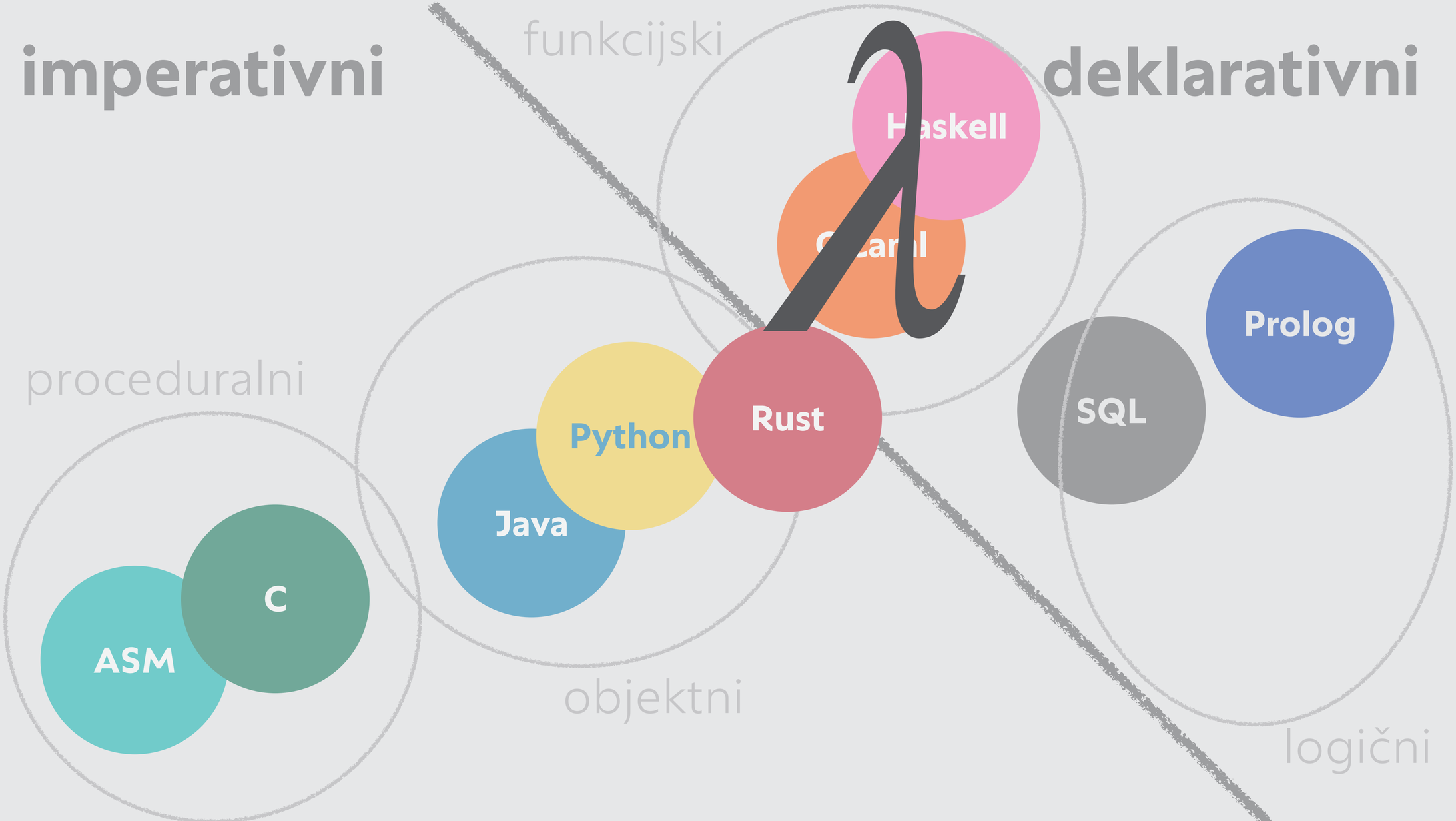
$$3^2 + 3 \cdot 3 + 7 \equiv 25$$

Spoznali smo **λ-račun**, ki vsebuje samo funkcije

izraz $e ::= x \mid \lambda x\,.\,e \mid e_1\,e_2$

abstrakcija aplikacija

λ-račun je temelj **funkcijskega programiranja**



Pogledali smo, kako različne konstrukte **predstavimo s funkcijami**

$$\underline{0} = \lambda f\, x\, .\, x$$

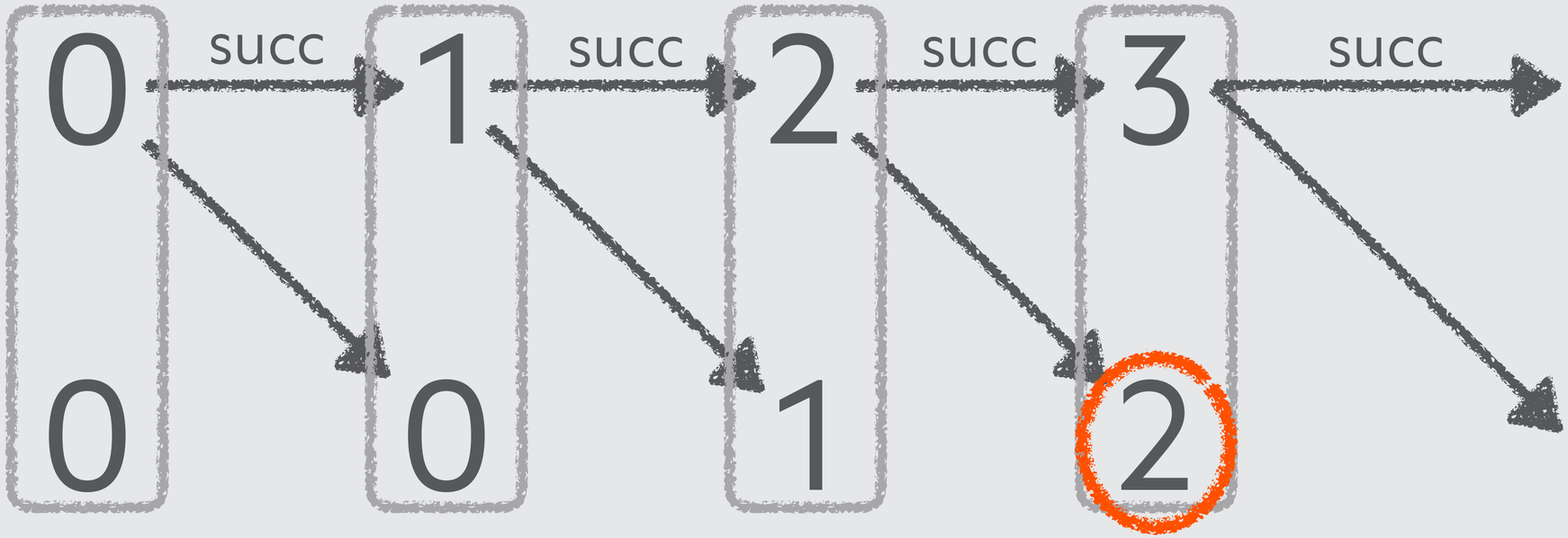
$$\underline{1} = \lambda f\, x\, .\, f\, x$$

$$\underline{2} = \lambda f\, x\, .\, f\, (f\, x)$$

$$\underline{n} = \lambda f\, x\, .\, \underbrace{f\, (f\, (\cdots (f\, x) \cdots))}_{n}$$

Alonzo Church, 1903–1995

Taka predstavitev podatkov **ni najbolj uporabna**



$$\lambda n \,.\, \text{second}\ \Big(n\ \big(\lambda p \,.\, \text{pair}\ (\text{succ}\ (\text{first}\ p))\ (\text{first}\ p)\big)\ \big(\text{pair}\ \underline{0}\ \underline{0}\big)\Big)$$

Konstrukcije množic

Kartezični **produkt** množic je množica vseh **urejenih parov**

$$A \times B = \{(x, y) \mid x \in A, y \in B\}$$

Velikost produkta končnih množic je **produkt velikosti**

$$|\{1,2,3\} \times \{4,5\}|$$

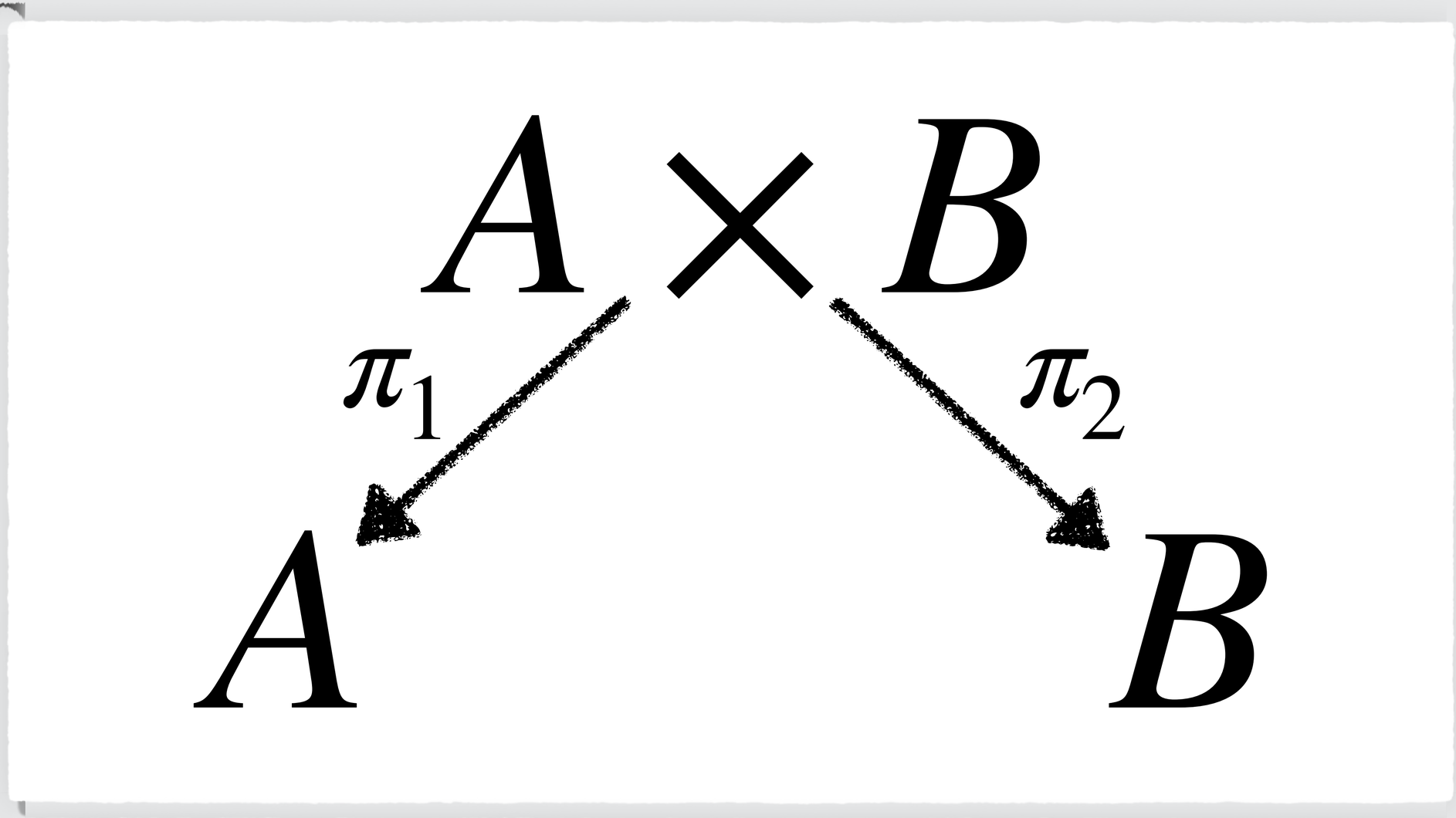
$$= |\{(1,4), (2,4), (3,4), (1,5), (2,5), (3,5)\}|$$

$$= |\{1,2,3\}| \cdot |\{4,5\}|$$

Produkti omogočajo, da hkrati delamo z **več vrednostmi**

$$\mathrm{maxLastna} : \mathbb{R}^{n^2} \to \mathbb{R} \times \mathbb{N}$$

Komponente parov dobimo prek **projekcij**



Kakšna konstrukcija bi bila **vsota množic?**

$$A + B$$

Velikost unije množic ni nujno **vsota velikosti**

$$|\{1,2,3\} \cup \{4,5\}|$$
$$= |\{1,2,3\}| + |\{4,5\}|$$

$$|\{1,2,3\} \cup \{3,4\}|$$
$$\neq |\{1,2,3\}| + |\{3,4\}|$$

Z **oznakami** dosežemo **disjunktnost** unije

$$\{1,2,3\} \cap \{3,4\} = \{3\}$$

$$\{\iota_1(1), \iota_1(2), \iota_1(3)\} \cap \{\iota_2(3), \iota_2(4)\} = \varnothing$$

Vsota dveh množic je njuna disjunktna unija

$$A + B = \{\iota_1(x) \mid x \in A\} \cup \{\iota_2(y) \mid y \in B\}$$

Velikost vsote končnih množic je **vsota velikosti**

$$|\{1,2,3\} + \{3,4\}|$$

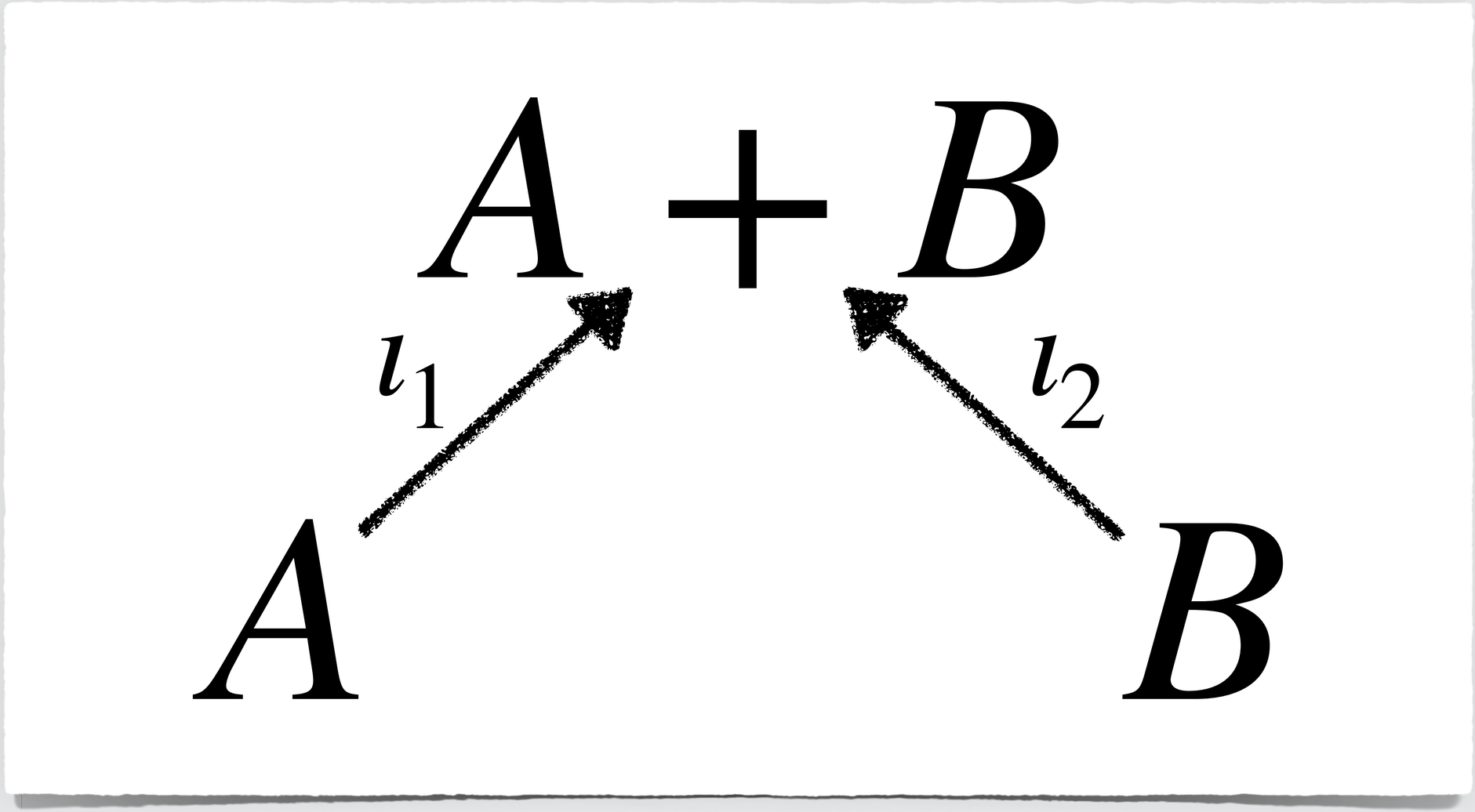
$$= |\{\iota_1(1), \iota_1(2), \iota_1(3)\} \cup \{\iota_2(3), \iota_2(4)\}|$$

$$= |\{1,2,3\}| + |\{3,4\}|$$

Vsote omogočajo, da hkrati zajememo **različne primere**

$$\mathrm{preberiCelo} : \Sigma^* \to \mathbb{Z} + \Sigma^*$$

V vsoto slikamo z dvema **injekcijama**



Iz vsote slikamo z **obravnavo primerov**

$$\text{vNaravno} : \mathbb{Z} + \Sigma^* \to \mathbb{N}$$

$$\text{vNaravno}(x) = \begin{cases} |n| & x = \iota_1(n) \\ 42 & x = \iota_2(s) \end{cases}$$

Kakšna konstrukcija bi bil **eksponent množic?**

$$B^A$$

Število funkcij med končnimi množicami je **eksponent** velikosti

$$\{1,2,3\} \to \{4,5\}$$

$$1 \mapsto 4 \quad 2 \mapsto 4 \quad 3 \mapsto 4$$

$$1 \mapsto 4 \quad 2 \mapsto 4 \quad 3 \mapsto 5$$

$$1 \mapsto 4 \quad 2 \mapsto 5 \quad 3 \mapsto 4$$

$$1 \mapsto 4 \quad 2 \mapsto 5 \quad 3 \mapsto 5$$

$$1 \mapsto 5 \quad 2 \mapsto 4 \quad 3 \mapsto 4$$

$$1 \mapsto 5 \quad 2 \mapsto 4 \quad 3 \mapsto 5$$

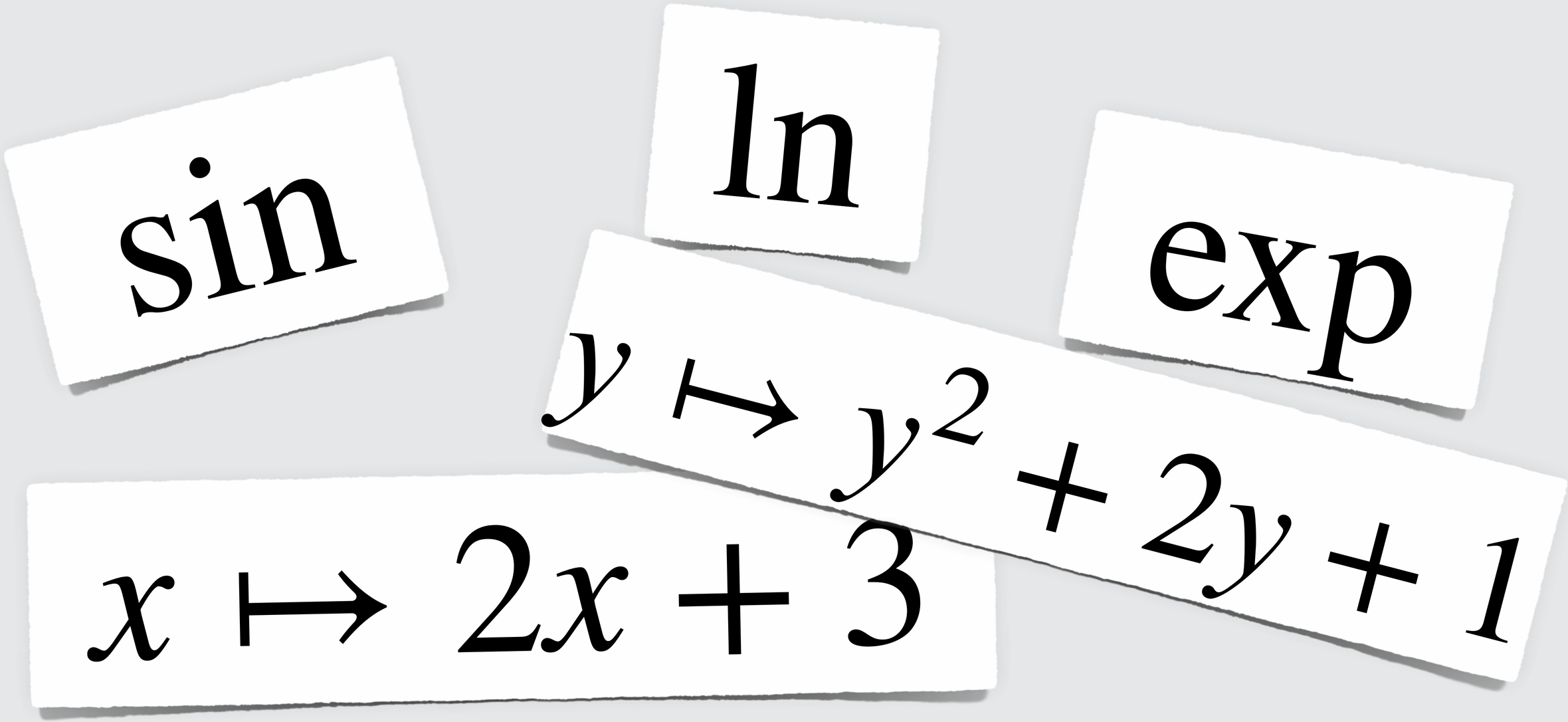
$$1 \mapsto 5 \quad 2 \mapsto 5 \quad 3 \mapsto 4$$

$$1 \mapsto 5 \quad 2 \mapsto 5 \quad 3 \mapsto 5$$

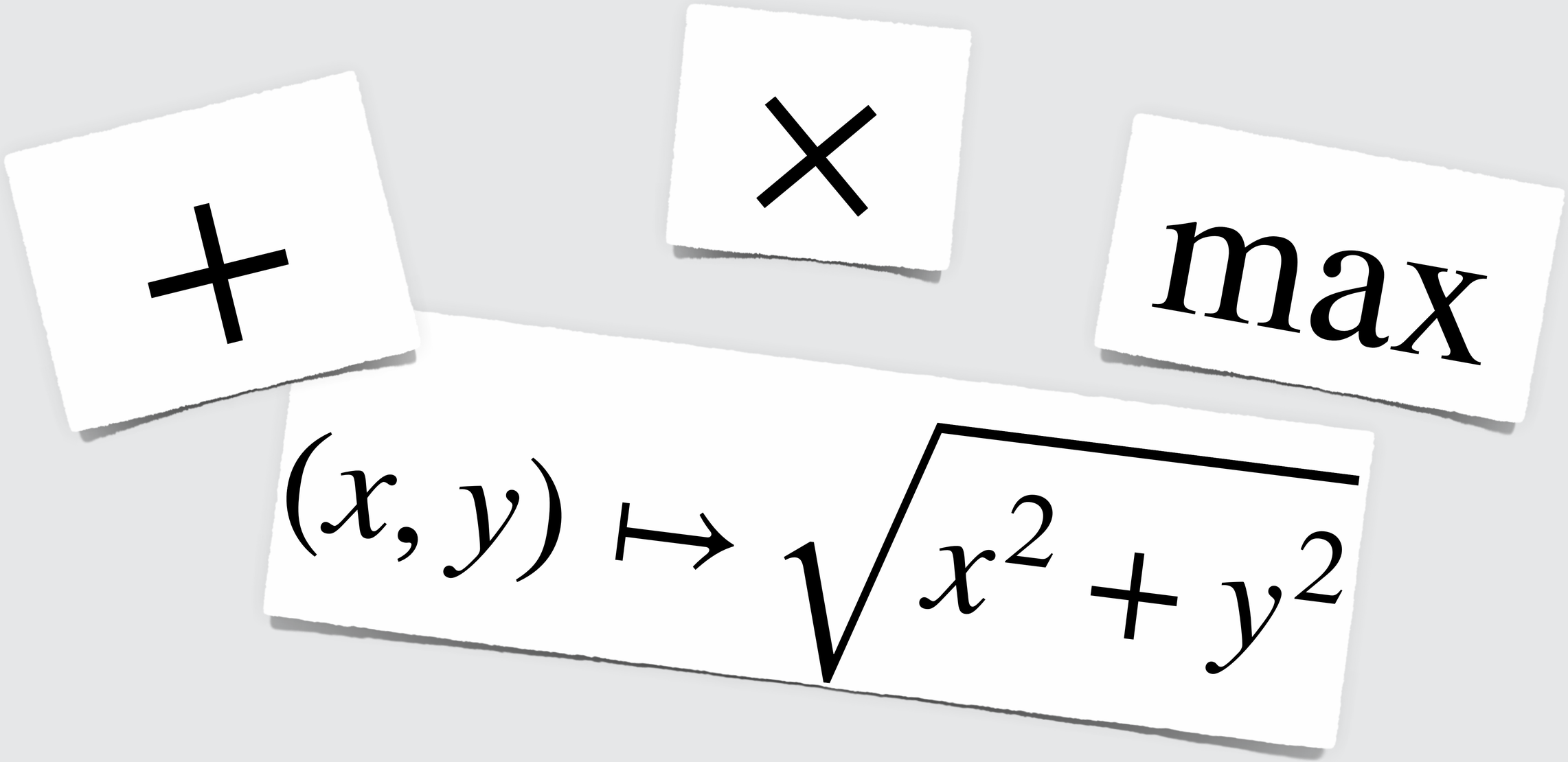
Eksponent dveh množic je množica **vseh preslikav** med njima

$$B^A = A \to B = \{f \mid f : A \to B\}$$

Eksponent $\mathbb{R} \to \mathbb{R}$ vsebuje **realne funkcije** ene spremenljivke



Eksponent $\mathbb{R}\times\mathbb{R}\to\mathbb{R}$ vsebuje funkcije **dveh spremenljivk**



Tudi na $\mathbb{R} \to (\mathbb{R} \to \mathbb{R})$ gledamo kot na funkcije **dveh spremenljivk**

$$k \mapsto (x \mapsto k \cdot x)$$

$$x \mapsto (y \mapsto \sqrt{x^2 + y^2})$$

Eksponent $(\mathbb{R} \to \mathbb{R}) \to \mathbb{R}$ vsebuje funkcije **drugega reda** (oz. funkcionale)

$$f \mapsto f(0)$$

$$f \mapsto \int_0^1 f(x)\,dx$$

Konstrukcije zadoščajo poprej videnim **izomorfizmom**

$$C^{A \times B} \cong (C^A)^B$$

$$(A + B) \times C \cong A \times C + B \times C$$

$$C^{A+B} \cong C^A \times C^B$$

Podatkovni tipi

Konstrukcije množic bi radi prenesli na **podatkovne tipe**

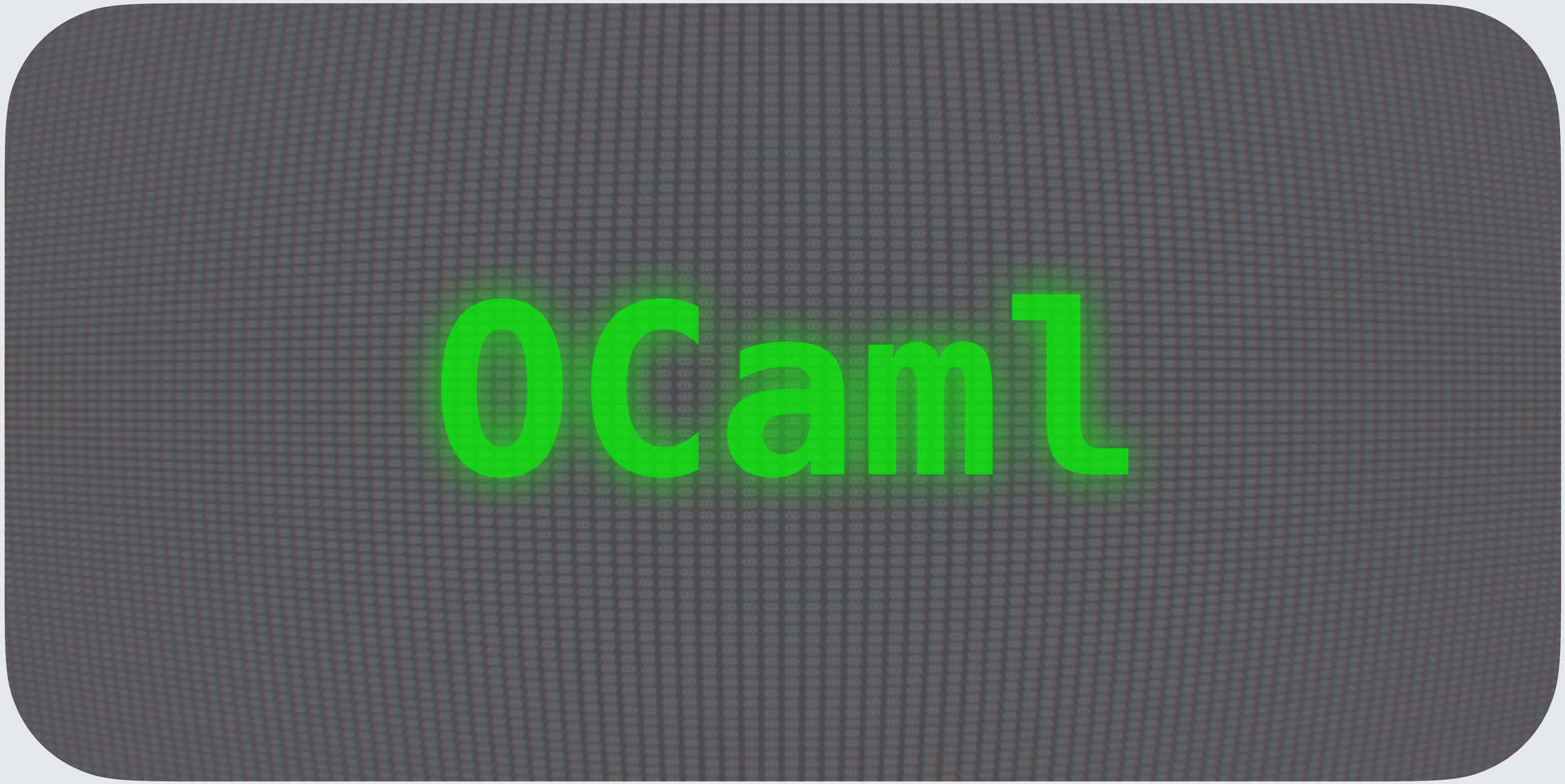
$$A \times B$$

$$A + B$$

$$A \to B$$

Konstrukcije tipov si bomo pogledali na primeru jezika **OCaml**

OCaml


OCaml

Osnove OCamla

Naš **prvi program** v OCamlu

In [1]:
```
let odgovor =
  let delni_izracun = max 2 3 + 4 in
  delni_izracun * 6
```

Out[1]:
```
val odgovor : int = 42
```

- vrednosti definiramo z `let`
- lokalne vrednosti definiramo z `let ... in ...`
- argumente funkcij lahko pišemo brez oklepajev
- uporaba (aplikacija) funkcij ima najvišjo prioriteto
- poleg vrednosti OCaml izračuna tudi tip programa

In [2]:
```
delni_izracun
```

```
File "[2]", line 1, characters 0-13:
1 | delni_izracun
    ^^^^^^^^^^^^^
Error: Unbound value delni_izracun
```

Celim številom priredimo tip `int`

In [3]:
```
12 * (34 + 67) - 89
```

Out[3]:
```
- : int = 1123
```

In [4]:
```
1024 / 100
```

```
Out[4]: - : int = 10
```

```
In [5]: 1024 mod 100
```

```
Out[5]: - : int = 24
```

Številom s plavajočo vejico priredimo tip `float`

```
In [6]: 12.0 *. (34.0 +. 67.0) -. 89.0
```

```
Out[6]: - : float = 1123.
```

```
In [7]: 1024. /. 100.
```

```
Out[7]: - : float = 10.24
```

```
In [8]: sqrt 2.
```

```
Out[8]: - : float = 1.41421356237309515
```

OCaml **strogo** ločuje med tipoma `int` in `float`

```
In [9]: 2. *. 3.141592
```

```
Out[9]: - : float = 6.283184
```

```
In [10]: float_of_int 10
```

```
Out[10]: - : float = 10.
```

```
In [11]: int_of_float 3.141592
```

```
Out[11]: - : int = 3
```

Logičnim vrednostim priredimo tip `bool`

In [12]:
```
3 <= 8
```

Out[12]:
```
- : bool = true
```

In [13]:
```
3 <= 8 && 8 <= 6
```

Out[13]:
```
- : bool = false
```

In [14]:
```
let abs x =
  if x < 0 then -x else x
```

Out[14]:
```
val abs : int -> int = <fun>
```

Nizom priredimo tip `string`

In [15]:
```
let fun_prog = "Funkcijsko programiranje"
```

Out[15]:
```
val fun_prog : string = "Funkcijsko programiranje"
```

Funkcije za delo z nizi se nahajajo v **modulu** `String`

In [16]:
```
String.length fun_prog
```

Out[16]:
```
- : int = 24
```

In [17]:
```
String.cat "Uvod v " (String.lowercase_ascii fun_prog)
```

Out[17]:
```
- : string = "Uvod v funkcijsko programiranje"
```

In [18]:
```
"Uvod v " ^ String.lowercase_ascii fun_prog
```

```
Out[18]: - : string = "Uvod v funkcijsko programiranje"
```

Enotski tip `unit` vsebuje samo eno vrednost

```
In [19]: ()
Out[19]: - : unit = ()
```

Tip `unit` uporabljamo v funkcijah, ki sprožajo **stranske učinke**.

```
In [20]: Random.int 100
Out[20]: - : int = 44
In [21]: Random.bool ()
Out[21]: - : bool = false
In [22]: print_endline "Hello, world!"
Hello, world!
Out[22]: - : unit = ()
```

Posameznim **znakom** priredimo tip `char`

```
In [23]: 'a'
Out[23]: - : char = 'a'
In [24]: Char.code 'x'
Out[24]: - : int = 120
```

In [25]:
```
Char.code 'y'
```

Out[25]:
```
- : int = 121
```

In [26]:
```
Char.chr 122
```

Out[26]:
```
- : char = 'z'
```

Primer: **Cezarjeva šifra**

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ    VENI, VIDI, VICI
KLMNOPQRSTUVWXYZABCDEFGHIJ    FOXS, FSNS, FSMS
```

In [27]:
```
let zamakni_znak zamik znak =
  if 'A' <= znak && znak <= 'Z' then
    let mesto_znaka = Char.code znak - Char.code 'A' in
    let novo_mesto = (mesto_znaka + zamik) mod 26 in
    Char.chr (Char.code 'A' + novo_mesto)
  else
    znak
```

Out[27]:
```
val zamakni_znak : int -> char -> char = <fun>
```

In [28]:
```
zamakni_znak 10 'R'
```

Out[28]:
```
- : char = 'B'
```

Primer: **Cezarjeva šifra**


```
ABCDEFGHIJKLMNOPQRSTUVWXYZ    VENI, VIDI, VICI
KLMNOPQRSTUVWXYZABCDEFGHIJ    FOXS, FSNS, FSMS
```

In [29]:
```
let sifriraj_niz zamik niz =
    let sifriraj_znak znak = zamakni_znak zamik znak in
    String.map sifriraj_znak niz
```

Out[29]:
```
val sifriraj_niz : int -> string -> string = <fun>
```

In [30]:
```
sifriraj_niz 10 "VENI, VIDI, VICI"
```

Out[30]:
```
- : string = "FOXS, FSNS, FSMS"
```

In [31]:
```
sifriraj_niz (26 - 10) "FOXS, FSNS, FSMS"
```

Out[31]:
```
- : string = "VENI, VIDI, VICI"
```

Funkcijski tipi

Funkcijam priredimo tip oblike $\texttt{tip}_{\texttt{arg}}\ \texttt{->}\ \texttt{tip}_{\texttt{rez}}$

In [32]:
```
float_of_int
```

Out[32]:
```
- : int -> float = <fun>
```

In [33]:
```
int_of_float
```

Out[33]:
```
- : float -> int = <fun>
```

In [34]:
```
String.length
```

Out[34]:
```
- : string -> int = <fun>
```

In [35]:
```
print_endline
```

Out[35]:
```
- : string -> unit = <fun>
```

Funkcije imajo lahko tudi **več argumentov**

In [36]:
```
let zmnozi x y = x * y
```

Out[36]:
```
val zmnozi : int -> int -> int = <fun>
```

In [37]:
```
String.cat
```

Out[37]:
```
- : string -> string -> string = <fun>
```

In [38]:
```
let zamakni_znak zamik znak =
  if 'A' <= znak && znak <= 'Z' then
    let mesto_znaka = Char.code znak - Char.code 'A' in
    let novo_mesto = (mesto_znaka + zamik) mod 26 in
    Char.chr (Char.code 'A' + novo_mesto)
  else
    znak
```

Out[38]:
```
val zamakni_znak : int -> char -> char = <fun>
```

Funkcije z več argumenti lahko tudi **delno uporabimo**

In [39]:
```
let sifriraj_niz zamik niz =
  let sifriraj_znak znak = zamakni_znak zamik znak in
  String.map sifriraj_znak niz
```

Out[39]:
```
val sifriraj_niz : int -> string -> string = <fun>
```

In [40]:
```
let rot13 = sifriraj_niz 13
```

Out[40]:
```
val rot13 : string -> string = <fun>
```

In [41]:
```
rot13 "VENI, VIDI, VICI"
```

Out[41]:
```
- : string = "IRAV, IVQV, IVPV"
```

In [42]:
```
rot13 "IRAV, IVQV, IVPV"
```

Out[42]:
```
- : string = "VENI, VIDI, VICI"
```

Delna uporaba omogoča precej **krajše programe**

In [43]:
```
let sifriraj_niz zamik niz =
  let sifriraj_znak znak = zamakni_znak zamik znak in
  String.map sifriraj_znak niz
```

Out[43]:
```
val sifriraj_niz : int -> string -> string = <fun>
```

In [44]:
```
let sifriraj_niz zamik niz =
  let sifriraj_znak = zamakni_znak zamik in
  String.map sifriraj_znak niz
```

Out[44]:
```
val sifriraj_niz : int -> string -> string = <fun>
```

In [45]:
```
let sifriraj_niz zamik niz =
  String.map (zamakni_znak zamik) niz
```

Out[45]:
```
val sifriraj_niz : int -> string -> string = <fun>
```

In [46]:
```
let sifriraj_niz zamik =
  String.map (zamakni_znak zamik)
```

Out[46]:
```
val sifriraj_niz : int -> string -> string = <fun>
```

Funkcije **višjega reda** za argumente sprejemajo druge funkcije

In [47]:
```
let je_samoglasnik znak =
  String.contains "aeiou" (Char.lowercase_ascii znak)
```

Out[47]:
```
val je_samoglasnik : char -> bool = <fun>
```

In [48]:
```
je_samoglasnik 'A'
```

Out[48]:
```
- : bool = true
```

In [49]:
```
let vsebuje_samoglasnik =
  String.exists je_samoglasnik
```

Out[49]:
```
val vsebuje_samoglasnik : string -> bool = <fun>
```

In [50]:
```
vsebuje_samoglasnik "čmrlj"
```

Out[50]:
```
- : bool = false
```

In [51]:
```
String.exists
```

Out[51]:
```
- : (char -> bool) -> string -> bool = <fun>
```

V funkcijskih tipih so **oklepaji pomembni**

In [52]:
```
String.make
```

Out[52]:
```
- : int -> char -> string = <fun>
```

In [53]:
```
String.make 10 '*'
```

Out[53]:
```
- : string = "**********"
```

In [54]:
```
String.init 10
```

Out[54]:
```
- : (int -> char) -> string = <fun>
```

In [55]:
```
let crka_abecede n = Char.chr (Char.code 'a' + n)
```

```
Out[55]: val crka_abecede : int -> char = <fun>
```

```
In [56]: String.init 26 crka_abecede
```

```
Out[56]: - : string = "abcdefghijklmnopqrstuvwxyz"
```

Kratke funkcije lahko pišemo tudi **anonimno**

```
In [57]: let zrcali niz =
           let n = String.length niz in
           let znak_na_zrcalnem_mestu i = String.get niz (n - i - 1) in
           String.init n znak_na_zrcalnem_mestu
```

```
Out[57]: val zrcali : string -> string = <fun>
```

```
In [58]: zrcali "perica reze raci rep"
```

```
Out[58]: - : string = "per icar ezer acirep"
```

```
In [59]: let zrcali niz =
           let n = String.length niz in
           String.init n (fun i -> String.get niz (n - i - 1))
```

```
Out[59]: val zrcali : string -> string = <fun>
```

```
In [60]: let je_stevilo niz =
           String.for_all (fun znak -> '0' <= znak && znak <= '9') niz
```

```
Out[60]: val je_stevilo : string -> bool = <fun>
```

Sestavljeni tipi

Več vrednosti istega tipa združujemo v **sezname**

In [61]:
```
[1; 2; 3; 4]
```

Out[61]:
```
- : int list = [1; 2; 3; 4]
```

In [62]:
```
['a'; 'b'; 'c'; 'd']
```

Out[62]:
```
- : char list = ['a'; 'b'; 'c'; 'd']
```

In [63]:
```
[1; 2; 3] @ [4; 5; 6]
```

Out[63]:
```
- : int list = [1; 2; 3; 4; 5; 6]
```

In [64]:
```
String.split_on_char ' ' "Uvod v funkcijsko programiranje"
```

Out[64]:
```
- : string list = ["Uvod"; "v"; "funkcijsko"; "programiranje"]
```

Najkoristnejše funkcije za delo s seznami so v modulu `List`

In [65]:
```
List.map String.length ["Uvod"; "v"; "funkcijsko"; "programiranje"]
```

Out[65]:
```
- : int list = [4; 1; 10; 13]
```

In [66]:
```
List.filter (fun x -> x < 5) [3; 1; 4; 1; 5; 9; 2; 6; 5; 3; 5; 9]
```

Out[66]:
```
- : int list = [3; 1; 4; 1; 2; 3]
```

In [67]:
```
List.flatten [[1; 2; 3]; [4; 5; 6]; [7; 8; 9]]
```

Out[67]:
```
- : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9]
```

Vrednosti **poljubnih** tipov združujemo v **nabore**

In [68]:
```
(25, "junij", 1991)
```

Out[68]:
```
- : int * string * int = (25, "junij", 1991)
```

In [69]:
```
[(1000, "Ljubljana"); (2000, "Maribor"); (3000, "Celje")]
```

Out[69]:
```
- : (int * string) list =
[(1000, "Ljubljana"); (2000, "Maribor"); (3000, "Celje")]
```

In [70]:
```
List.partition (fun x -> x < 5) [3; 1; 4; 1; 5; 9; 2; 6; 5; 3; 5; 9]
```

Out[70]:
```
- : int list * int list = ([3; 1; 4; 1; 2; 3], [5; 9; 6; 5; 5; 9])
```

Nabore lahko **razstavljamo** z vzorci

In [71]:
```
let raztegni faktor koord =
  (faktor *. fst koord, faktor *. snd koord)
```

Out[71]:
```
val raztegni : float -> float * float -> float * float = <fun>
```

In [72]:
```
let raztegni faktor koord =
  let (x, y) = koord in
  (faktor *. x, faktor *. y)
```

Out[72]:
```
val raztegni : float -> float * float -> float * float = <fun>
```

In [73]:
```
let raztegni faktor (x, y) =
  (faktor *. x, faktor *. y)
```

Out[73]:
```
val raztegni : float -> float * float -> float * float = <fun>
```

Vrednosti lahko **ignoriramo** z vzorcem `_`

In [74]:
```
let poste = [(1000, "Ljubljana"); (2000, "Maribor"); (3000, "Celje")]
```

Out[74]:
```
val poste : (int * string) list =
  [(1000, "Ljubljana"); (2000, "Maribor"); (3000, "Celje")]
```

In [75]:
```
List.split poste
```

Out[75]:
```
- : int list * string list =
([1000; 2000; 3000], ["Ljubljana"; "Maribor"; "Celje"])
```

In [76]:
```
let (_, imena_krajev) = List.split poste
```

Out[76]:
```
val imena_krajev : string list = ["Ljubljana"; "Maribor"; "Celje"]
```

Za **delno definirane** funkcije uporabljamo tip `option`

In [77]:
```
int_of_string "100"
```

Out[77]:
```
- : int = 100
```

In [78]:
```
int_of_string "sto"
```

```
Exception: Failure "int_of_string".
Raised by primitive operation at unknown location
Called from Stdlib__Fun.protect in file "fun.ml", line 33, characters 8-15
Re-raised at Stdlib__Fun.protect in file "fun.ml", line 38, characters 6-52
Called from Topeval.load_lambda in file "toplevel/byte/topeval.ml", line 89, characters 4-150
```

In [ ]:
```
int_of_string_opt "100"
```

In [ ]:
```
int_of_string_opt
```

In [ ]:
```
List.filter_map int_of_string_opt ["100"; "sto"; "123"; "tisoč"]
```

Polimorfni tipi

Kakšen je tip lepljenja seznamov?

```
In [79]: [true; false] @ [false; true]
Out[79]: - : bool list = [true; false; false; true]

In [80]: [1; 2] @ [3; 4; 5]
Out[80]: - : int list = [1; 2; 3; 4; 5]

In [81]: ( @ )
Out[81]: - : 'a list -> 'a list -> 'a list = <fun>
```

Vrednostim, ki se pri vseh tipih obnašajo enako, pravimo **parametrično polimorfne**

```
In [82]: List.flatten
Out[82]: - : 'a list list -> 'a list = <fun>

In [83]: List.filter
Out[83]: - : ('a -> bool) -> 'a list -> 'a list = <fun>

In [84]: []
Out[84]: - : 'a list = []
```

V polimorfnih tipih lahko nastopa **več parametrov**

```
In [85]: snd
Out[85]: - : 'a * 'b -> 'b = <fun>

In [86]: List.map
```

```
Out[86]: - : ('a -> 'b) -> 'a list -> 'b list = <fun>
```

```
In [87]: List.filter_map
```

```
Out[87]: - : ('a -> 'b option) -> 'a list -> 'b list = <fun>
```

```
In [88]: List.combine
```

```
Out[88]: - : 'a list -> 'b list -> ('a * 'b) list = <fun>
```

Veriženje

Funkcije **verižimo** z operacijo `|>`

```
In [89]: let ( |> ) x f = f x
```

```
Out[89]: val ( |> ) : 'a -> ('a -> 'b) -> 'b = <fun>
```

```
In [90]: String.length (String.trim "   beseda   ")
```

```
Out[90]: - : int = 6
```

```
In [91]: "   beseda   " |> String.trim |> String.length
```

```
Out[91]: - : int = 6
```

```
In [92]: "100,sto,123,tisoč"
         |> String.split_on_char ','
         |> List.filter_map int_of_string_opt
```

```
Out[92]: - : int list = [100; 123]
```

Primer: **izločanje števil** v besedilu

```
1000,Ljubljana          [[1000];
sto,100           ~>     [100];
1,a,2,b,3                [1; 2; 3]]
```

In [93]:
```
let stevila_v_vrstici vrstica =
  vrstica |> String.split_on_char ',' |> List.filter_map int_of_string_opt
```

Out[93]:
```
val stevila_v_vrstici : string -> int list = <fun>
```

In [94]:
```
let stevila_v_besedilu besedilo =
  besedilo |> String.split_on_char '\n' |> List.map stevila_v_vrstici
```

Out[94]:
```
val stevila_v_besedilu : string -> int list list = <fun>
```

In [95]:
```
"1000,Ljubljana
sto,100
1,a,2,b,3
" |> stevila_v_besedilu
```

Out[95]:
```
- : int list list = [[1000]; [100]; [1; 2; 3]; []]
```

Ujemanje vzorcev

Funkcijo **po kosih** definiramo z `match`

In [96]:
```
let ime_jezika koncnica =
  if koncnica = ".ml" then
    "OCaml"
  else if koncnica = ".py" then
    "Python"
  else if koncnica = ".rs" then
    "Rust"
  else
    "???"
```

Out[96]:
```
val ime_jezika : string -> string = <fun>
```

In [97]:
```
let ime_jezika koncnica =
  match koncnica with
  | ".ml" -> "OCaml"
  | ".py" -> "Python"
  | ".rs" -> "Rust"
  | _ -> "???"
```

Out[97]:
```
val ime_jezika : string -> string = <fun>
```

In [98]:
```
ime_jezika ".java"
```

Out[98]:
```
- : string = "???"
```

Za funkcije, ki **takoj** izvedejo `match`, raje uporabimo `function`

In [99]:
```
let ime_jezika koncnica =
  match koncnica with
  | ".ml" -> "OCaml"
  | ".py" -> "Python"
  | ".rs" -> "Rust"
  | _ -> "???"
```

Out[99]:
```
val ime_jezika : string -> string = <fun>
```

In [100…
```
let ime_jezika =
  function
  | ".ml" -> "OCaml"
  | ".py" -> "Python"
  | ".rs" -> "Rust"
  | _ -> "???"
```

Out[100]:
```
val ime_jezika : string -> string = <fun>
```

Ujemanje izvede **prvo** vejo, pri kateri se vrednost **ujema z vzorcem**

In [101…
```
let ime_jezika =
  function
  | ".ml" -> "OCaml"
  | ".py" -> "Python"
  | ".rs" -> "Rust"
   | _ -> "???"
```

Out[101]:
```
val ime_jezika : string -> string = <fun>
```

In [102…
```
ime_jezika ".rs"
```

Out[102]:
```
- : string = "Rust"
```

Zajeti moramo **vse** vzorce

In [103…
```
let ime_jezika =
  function
  | ".ml" -> "OCaml"
  | ".py" -> "Python"
  | ".rs" -> "Rust"
  | "" -> "prazen jezik"
```

```
File "[103]", lines 2-6, characters 2-24:
2 | ..function
3 |   | ".ml" -> "OCaml"
4 |   | ".py" -> "Python"
5 |   | ".rs" -> "Rust"
6 |   | "" -> "prazen jezik"
Warning 8 [partial-match]: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
"*"
```

Out[103]: `val ime_jezika : string -> string = <fun>`

In [104…
```
ime_jezika ".java"
```

```
Exception: Match_failure ("[103]", 2, 2).
Raised at ime_jezika in file "[103]", line 2, characters 2-98
Called from Stdlib__Fun.protect in file "fun.ml", line 33, characters 8-15
Re-raised at Stdlib__Fun.protect in file "fun.ml", line 38, characters 6-52
Called from Topeval.load_lambda in file "toplevel/byte/topeval.ml", line 89, characters 4-150
```

Vsaka **konstanta** je vzorec

In [105…
```
let ali_je_res =
  function
  | true -> "res je"
  | false -> "ni res"
```

Out[105]: `val ali_je_res : bool -> string = <fun>`

In [106…
```
let ime_stevila =
  function
  | 1 -> "ena"
  | 2 -> "dva"
  | _ -> "veliko"
```

Out[106]: `val ime_stevila : int -> string = <fun>`

Spremenljivka je vzorec, ki se ujema z **vsemi** vrednosti, ter v veji omogoči **dostop do zajete vrednosti**

In [107…
```
let ime_stevila =
  function
  | 1 -> "ena"
  | 2 -> "dva"
  | x -> "število " ^ string_of_int x
```

Out[107]:
```
val ime_stevila : int -> string = <fun>
```

In [108…
```
ime_stevila 5
```

Out[108]:
```
- : string = "število 5"
```

In [109…
```
ime_stevila 3
```

Out[109]:
```
- : string = "število 3"
```

Več zaporednih vzorcev lahko tudi združimo

In [110…
```
let je_prestopno leto =
  (leto mod 4 = 0 && leto mod 100 <> 0) || leto mod 400 = 0

let dolzina_meseca leto =
  function
  | 4 | 6 | 9 | 11 -> 30
  | 2 -> if je_prestopno leto then 29 else 28
  | _ -> 31
```

Out[110]:
```
val je_prestopno : int -> bool = <fun>
```

Out[110]:
```
val dolzina_meseca : int -> int -> int = <fun>
```

Sestavljeni vzorci

Vzorec `(v1, v2, ...)` ustreza **naborom**,
pri čemer so `v1`, `v2`, ... **nadaljnji gnezdeni** vzorci

In [111…
```
let polozaj_tocke =
  function
  | (0, 0) -> "izhodišče"
  | (_, 0) -> "abscisa"
  | (0, _) -> "ordinata"
  | (_, _) -> "nekje drugje"
```

Out[111]:
```
val polozaj_tocke : int * int -> string = <fun>
```

V vsakem vzorcu se spremenljivka lahko **pojavi le enkrat**

In [112…
```
let polozaj_tocke =
  function
  | (0, 0) -> "izhodišče"
  | (_, 0) -> "abscisa"
  | (0, _) -> "ordinata"
  | (x, x) -> "diagonala"
  | (_, _) -> "nekje drugje"
```

```
File "[112]", line 6, characters 8-9:
6 |   | (x, x) -> "diagonala"
            ^
Error: Variable x is bound several times in this matching
```

In [113…
```
let polozaj_tocke =
  function
  | (0, 0) -> "izhodišče"
  | (_, 0) -> "abscisa"
  | (0, _) -> "ordinata"
  | (x, y) -> if x = y then "diagonala" else "nekje drugje"
```

Out[113]:
```
val polozaj_tocke : int * int -> string = <fun>
```

Vzorec `v when pogoj` se ujema le ob **izpolnjenem pogoju**

In [114…
```
let polozaj_tocke =
  function
  | (0, 0) -> "izhodišče"
  | (_, 0) -> "abscisa"
  | (0, _) -> "ordinata"
  | (x, y) -> if x = y then "diagonala" else "nekje drugje"
```

Out[114]:
```
val polozaj_tocke : int * int -> string = <fun>
```

In [115…
```
let polozaj_tocke =
  function
  | (0, 0) -> "izhodišče"
  | (_, 0) -> "abscisa"
  | (0, _) -> "ordinata"
  | (x, y) when x = y -> "diagonala"
  | (_, _) -> "nekje drugje"
```

Out[115]:
```
val polozaj_tocke : int * int -> string = <fun>
```

Zapisi

Zapisni tip podamo z zahtevanimi polji

In [116…
```
type datum = int * int * int
```

Out[116]:
```
type datum = int * int * int
```

In [117…
```
type datum = { dan : int; mesec : int; leto : int }
```

Out[117]:
```
type datum = { dan : int; mesec : int; leto : int; }
```

In [118…
```
let osamosvojitev = { dan = 25; mesec = 6; leto = 1991 }
```

Out[118]:
```
val osamosvojitev : datum = {dan = 25; mesec = 6; leto = 1991}
```

Do polj lahko dostopamo prek **projekcij**

In [119…
```
let je_prestopno leto =
  (leto mod 4 = 0 && leto mod 100 <> 0) || leto mod 400 = 0

let dolzina_meseca leto =
  function
  | 4 | 6 | 9 | 11 -> 30
  | 2 -> if je_prestopno leto then 29 else 28
  | _ -> 31
```

Out[119]:
```
val je_prestopno : int -> bool = <fun>
```

Out[119]:
```
val dolzina_meseca : int -> int -> int = <fun>
```

In [120…
```
let je_veljaven datum =
  let veljaven_dan = 1 <= datum.dan && datum.dan <= dolzina_meseca datum.leto datum.mesec
  and veljaven_mesec = 1 <= datum.mesec && datum.mesec <= 12
  in
  veljaven_dan && veljaven_mesec
```

Out[120]:
```
val je_veljaven : datum -> bool = <fun>
```

Zapise lahko **razstavljamo** tudi prek vzorcev

In [121…
```
let je_veljaven {dan = d; mesec = m; leto = l} =
  let veljaven_dan = 1 <= d && d <= dolzina_meseca l m
  and veljaven_mesec = 1 <= m && m <= 12
  in
  veljaven_dan && veljaven_mesec
```

Out[121]:
```
val je_veljaven : datum -> bool = <fun>
```

In [122…
```
let je_veljaven {dan; mesec; leto} =
  let veljaven_dan = 1 <= dan && dan <= dolzina_meseca leto mesec
  and veljaven_mesec = 1 <= mesec && mesec <= 12
  in
  veljaven_dan && veljaven_mesec
```

Out[122]:
```
val je_veljaven : datum -> bool = <fun>
```

Zapise lahko **posodabljamo** z `with`

In [123…
```
let pred_sto_leti datum =
  {dan = datum.dan; mesec = datum.mesec; leto = datum.leto - 100}
```

Out[123]:
```
val pred_sto_leti : datum -> datum = <fun>
```

In [124…
```
let pred_sto_leti datum =
  {datum with leto = datum.leto - 100}
```

Out[124]:
```
val pred_sto_leti : datum -> datum = <fun>
```

Naštevni tipi

Naštevni tip podamo z možnimi **variantami**

In [125…
```
type dostava =
  | OsebniPrevzem
  | PoPosti
```

Out[125]:
```
type dostava = OsebniPrevzem | PoPosti
```

Funkcije na naštevnih tipih podamo **po kosih**

In [126…
```
type dostava =
  | OsebniPrevzem
  | PoPosti
```

Out[126]:
```
type dostava = OsebniPrevzem | PoPosti
```

In [127…
```
let cena_dostave =
  function
  | OsebniPrevzem -> 0.
  | PoPosti -> 2.5
```

Out[127]:
```
val cena_dostave : dostava -> float = <fun>
```

Prevajalnik nas sam opozori na **manjkajoče primere**

In [128…
```
type dostava =
  | OsebniPrevzem
  | PoPosti
  | HitraDostava
```

Out[128]:
```
type dostava = OsebniPrevzem | PoPosti | HitraDostava
```

In [129…
```
let cena_dostave =
  function
  | OsebniPrevzem -> 0.
  | PoPosti -> 2.5
```

```
File "[129]", lines 2-4, characters 2-18:
2 | ..function
3 |   | OsebniPrevzem -> 0.
4 |   | PoPosti -> 2.5
Warning 8 [partial-match]: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
HitraDostava
```

Out[129]:
```
val cena_dostave : dostava -> float = <fun>
```

Konstruktorji lahko sprejmejo tudi **argumente**

In [130…
```
type naslov = string
type telefon = string
type dostava =
  | OsebniPrevzem
  | PoPosti of naslov
  | HitraDostava of naslov * telefon
```

Out[130]:
```
type naslov = string
```

Out[130]:
```
type telefon = string
```

Out[130]:
```
type dostava =
    OsebniPrevzem
  | PoPosti of naslov
  | HitraDostava of naslov * telefon
```

Tip `option` je primer **parametriziranega** naštevnega tipa

In [131…
```
type 'a option =
  | None
  | Some of 'a
```

Out[131]:
```
type 'a option = None | Some of 'a
```

Jezik nas prisili, da **obravnavamo manjkajoče** vrednosti

In [132…
```
let opisuje_pozitivno_stevilo niz =
  int_of_string niz > 0
```

Out[132]:
```
val opisuje_pozitivno_stevilo : string -> bool = <fun>
```

In [133…
```
opisuje_pozitivno_stevilo "sto"
```

```
Exception: Failure "int_of_string".
Raised by primitive operation at opisuje_pozitivno_stevilo in file "[132]", line 2, characters 2-19
Called from Stdlib__Fun.protect in file "fun.ml", line 33, characters 8-15
Re-raised at Stdlib__Fun.protect in file "fun.ml", line 38, characters 6-52
Called from Topeval.load_lambda in file "toplevel/byte/topeval.ml", line 89, characters 4-150
```

In [134…
```
let opisuje_pozitivno_stevilo niz =
  match int_of_string_opt niz with
  | Some stevilo -> stevilo > 0
  | None -> false
```

Out[134]:
```
val opisuje_pozitivno_stevilo : string -> bool = <fun>
```

prihodnjič...

Videli bomo, da **rekurzija** ustreza iskanju **negibnih točk**

Pogledali bomo **rekurzivne tipe**

```
type json =
   | String of string
   | Number of int
   | Object of (string * json) list
   | Array of json array
   | True
   | False
   | Null
```

Pogledali bomo **neskončne** rekurzivne tipe

```haskell
from :: Integer -> Stream Integer
from n = Cons n (from (n + 1))

sieve :: Stream Integer -> Stream Integer
sieve (Cons k s) =
    Cons k
      (sieve (filter (\n -> n `mod` k /= 0) s))

primes :: Stream Integer
primes = sieve $ from 2
```