PROGRAMIRANJE
2014/15

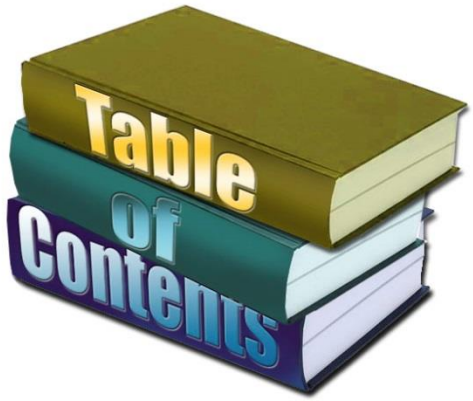
opcije

deklariranje podatkovnih tipov

ujemanje vzorcev

polimorfizem

Pregled



- sestavljeni podatkovni tipi
  - terke (angl. tuples)
  - seznami (angl. lists)
- vezave v lokalnem okolju
  - namen
  - odstranitev odvečnih parametrov
  - optimizacija rekurzivnih klicev
- podatkovni tip „opcija“ (angl. *option*)

Problem

- v premislek:
  - kateri je minimalni element praznega seznama?
  - katero je zaporedno mesto (pozicija v seznamu) podanega elementa, ki ga v seznamu ni?

- kaj vrniti kot odgovor?
  - -1 ?
  - [ ] ?
  - null ?
  - prožiti izjemo?

- rešitev v SML: opcija, vezana na podatkovni tip:
  - `SOME <rezultat>`, če rezultat obstaja
  - `NONE`, če rezultat ni veljaven

Opcije

- tip `t option` (npr. `int option, string option, …`)
  - podobno kot "`list`" v primerih: `int list`, `(int*bool) list` itd.

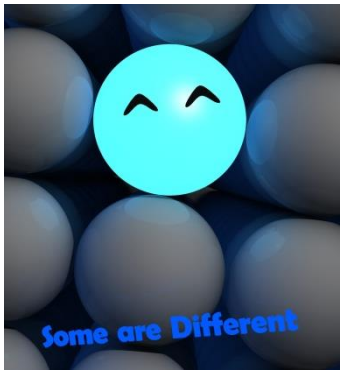
- zapis opcije
  - `SOME e` ➔ če je `e` tipa `t`, je `SOME e` tipa `t option`
  - `NONE` ➔ je tipa `'a option`

- dostop do opcije
  - `isSome`: preveri, ali je opcija v obliki `SOME`

```
val it = fn : 'a option -> bool
```

  - `valOf`: vrne vrednost e opcije `SOME e`

```
val it = fn : 'a option -> 'a
```



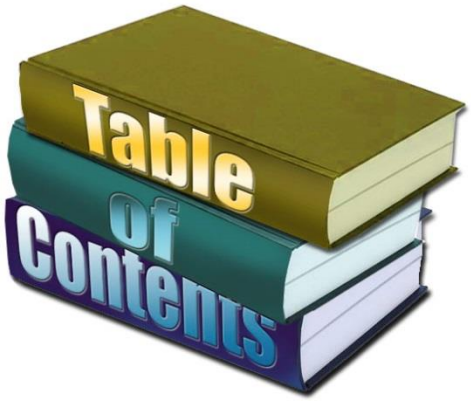
Izboljšava iskanja elementa

- notranje funkcije lahko uporabljajo zunanje vezave, odvečne (podvojene) reference lahko torej odstranimo

```
(* poiscemo prvo lokacijo pojavitve elementa el *)
(* (int list * int) -> int option *)

fun najdi(sez: int list, el: int) =
    if null sez
    then NONE
    else if (hd sez = el)
    then SOME 1
    else let val preostanek = najdi (tl sez, el)
         in if isSome preostanek
            then SOME (1+ valOf preostanek)
            else NONE
         end
```

Pregled



- sestavljeni podatkovni tipi
  - zapisi (angl. records)
- sinonimi za podatkovne tipe
- izdelava lastnih podatkovnih tipov
- ujemanje vzorcev s stavkom case
- definicija seznama in opcije
- polimorfizem podatkovnih tipov
- ujemanje vzorcev pri deklaracijah

Podatkovni tipi – do sedaj

- enostavni PT
  - `int`
  - `bool`
  - `real`
  - `string`
  - `char`

- sestavljeni (kompleksni) podatkovni tipi
  - terke `(e1, e2, …, en)` – tip `t1 * t2 * … *tn`
  - seznami `[e1, e2, …, en]` – tip `a' list`
  - opcije `SOME e, NONE` – tip `a' option`
  - zapisi

- izdelava lastnih podatkovnih tipov?

} tema za danes

Zapis (angl. *record*)

- podatkovni tip s **poljubnim** številom **imenovanih** polj, ki hranijo vrednosti (lahko različnih podatkovnih podtipov)

- zapis zapisa:

```
{polje1 = e1, polje2 = e2, …, poljen = en}
```

  - če je podatkovni tip komponent enak `e1: t1, …, en: tn`, ima celotni zapis podatkovni tip `{polje1: t1, …, poljen: tn}`
    - vrstni red polj ni pomemben (SML prikaže v abecednem vrstnem redu)
    - tipi so lahko enostavni ali sestavljeni
    - podani so lahko izrazi, ki se pri deklaraciji evalvirajo v vrednosti
    - SML implicitno deklarira novi tip zapisa (ni treba tega narediti nam)

- dostop do elementov zapisa e

```
#ime_polja e
```

Primer uporabe zapisa

```
val zapis = {ime="Dejan", starost=21, absolvent=false,
    ocene=[("angl",8),("ars",10)] }

#absolvent zapis;

#ocene zapis;
```

```
(#ime zapis) ^ " je star "
             ^ Int.toString(#starost zapis) ^ " let."
```

Sinonimi za podatkovne tipe

Pogosto uporabljene in kompleksne (dolge) nazive podatkovnih tipov lahko poimenujemo z lastnim imenom in si poenostavimo delo.

```
type novo_ime = tip
```

```
fun izpis_studenta (zapis: {absolvent:bool, ime:string,
                            ocene:(string * int) list, starost:int}) =
    (#ime zapis) ^ " je star " ^  Int.toString(#starost zapis) ^ " let."
```

```
type student = {absolvent:bool, ime:string,
                ocene:(string * int) list, starost:int}
```

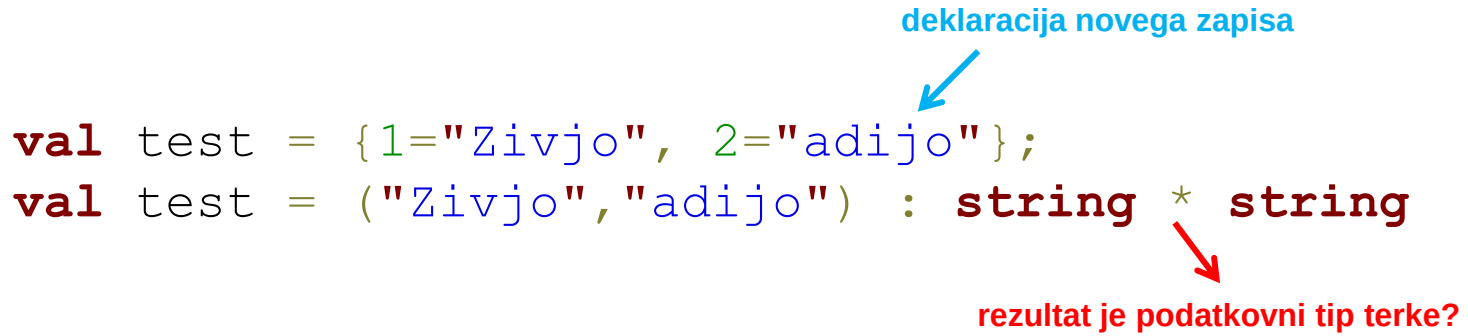
```
fun izpis_studenta2 (zapis: student) =
    (#ime zapis) ^ " je star " ^  Int.toString(#starost zapis) ^ " let."
```

- obe imeni tipov sta ekvivalentni
- SML lahko pri zapisovanju funkcij uporablja novo ali staro (dolgo) ime tipa (nepomembno)

```
val izpis_studenta2 = fn : student -> string
```

Terke in zapisi

- poglejmo si zanimiv primer…



- poseben tip "terka" torej v programskem jeziku ne obstaja! Terka je torej samo sintaktična olepšava/bližnjica za posebno obliko zapisa:
  - zapis (e1,…,en) namesto {1=e1,…,n=en}
  - zapis podatkovnega tipa t1*…*tn namesto {1:t1, …, n:tn}

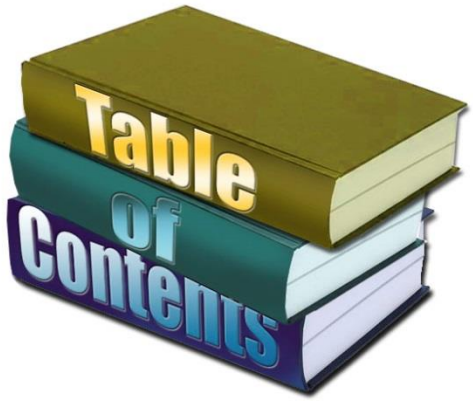
- sintaktična olepšave nam omogočajo lažje delo s programskim jezikom (razumevanje jezika in implementacijo lastnih programov)

Še več o terkah...

- terka – naslavljanje po vrstnem redu argumentov;
  zapis – naslavljanje po imenih argumentov
  - kdaj pri programiranju uporabljamo enega in drugega?

- terke ali polja?
  - pri majhnem številu elementov nam ni potrebno pomniti imen polj,
  - pri velikem številu elementov lažje pomnimo komponente po imenu kot po vrstnem redu

Pregled



- sestavljeni podatkovni tipi
  - zapisi (angl. records)
- sinonimi za podatkovne tipe
- izdelava lastnih podatkovnih tipov
- ujemanje vzorcev s stavkom case
- definicija seznama in opcije
- polimorfizem podatkovnih tipov
- ujemanje vzorcev pri deklaracijah

Izdelava lastnih podatkovnih tipov

- deklaracija novega podatkovnega tipa, ki predstavlja **alternativo** med podatkovnimi tipi, iz katerih je sestavljen:

```
datatype prevozno_sredstvo = Bus of int
                           | Avto of string * string
                           | Pes
```

konstruktorji

vsebina podatkovnega tipa

- (ali obstaja kaj podobnega v drugih programskih jezikih?)

- rezultat:
  - v okolju definiramo novi podatkovni tip `prevozno_sredstvo`
  - v okolju definiramo konstruktorje za izdelavo novih podatkovnih tipov: `Bus`, `Avto` in `Pes`

Vrednosti lastnih podatkovnih tipov

- vrednost novega podatkovnega tipa je vedno sestavljena z **oznako konstruktorja** (+ **vrednost**), npr:
  - `Bus 1`
  - `Avto ("fiat", "modri")`
  - `Pes`

- konstruktorja Bus in Avto sta funkciji, ki vrneta vrednost novega podatkovnega tipa:

```
fn : int -> prevozno_sredstvo
fn : string * string -> prevozno_sredstvo
```

- konstruktor Pes ne potrebuje argumenta in že sam predstavlja vrednost

```
val it = Pes : prevozno_sredstvo
```

- vrednost novega pod. tipa lahko opredelimo tudi z izrazom, npr. `Bus (1+5)`

Prednosti?

- omogoča definiranje različnih alternativ zapisa podatka
  - namesto redundantnih zapisov:

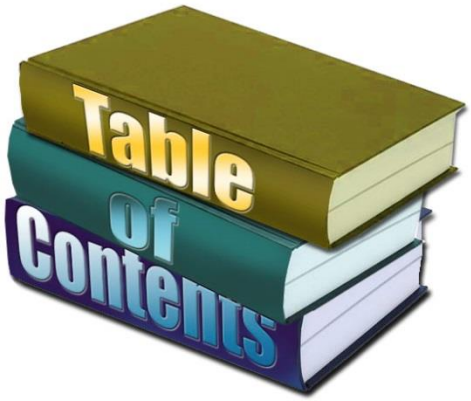
```
(* ce nacin =1 glej polje bus;
   ce = 2, glej avto; ce je 3 glej pes *)
 { nacin: int,
   bus: int,
   avto: string*string,
   pes: boolean}
```

ustvarimo eleganten (izključujoč) podatkovni tip

```
datatype prevozno_sredstvo = Bus of int
                           | Avto of string * string
                           | Pes
```

- omogoča rekurzivno definiranje tipa (pomembno za sezname, kasneje podrobno o tem…)

Pregled



- sestavljeni podatkovni tipi
  - zapisi (angl. records)
- sinonimi za podatkovne tipe
- izdelava lastnih podatkovnih tipov
- ujemanje vzorcev s stavkom case
- definicija seznama in opcije
- polimorfizem podatkovnih tipov
- ujemanje vzorcev pri deklaracijah

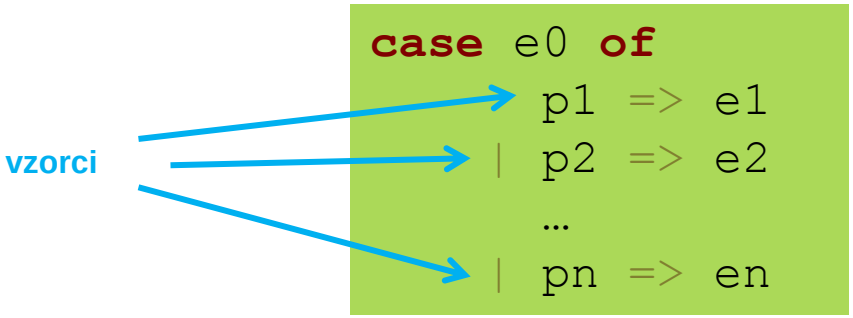
Delo z lastnimi podatkovnimi tipi

- lastni podatkovni tipi predstavljajo **alternativne** komponente

- (teoretično) imamo dve možnosti načina uporabe:
  1. pri programiranju **sproti preverjati**, s katerim podtipom dejansko delamo (ali je tip `prevozno_sredstvo` dejansko vrste `Bus`, `Avto` ali `Pes`?)
     - uporaba funkcij, kot bi bile `isBus`, `isAvto` (podobno kot `isSome` in `null`), in pridobiti podatke npr. z `getBusInt`, `getAvtoStrStr` (podobno kot `hd`, `tl`, `valOf`)
     - tak način je pogosto prisoten v dinamično tipiziranih jezikih (kako je s tem pri Javi?)
  2. podatek **primerjati z različnimi vzorci**
     - SML uporablja sistem primerjanja z vzorci!
     - stavek `case`

Stavek *case*

- primerja podani izraz `e0` za ujemanje z vzorci `p1, …, pn`
- rezultat je (samo eden) izraz na desni strani vzorca, s katerim se `e0` ujema
- vse veje `e1, …, en` morajo biti istega podatkovnega tipa



- primer:
  - naši vzorci so možne alternative podatkovnega tipa (konstruktor + spremenljivka)
  - spremenljivke v vzorcu dobijo dejanske vrednosti glede na podani argument

```
fun obdelaj_prevoz x =
    case x of
        Bus i => i+10
      | Avto (s1,s2) => String.size s1 + String.size s2
      | Pes => 0
```

Stavek *case*

- prednosti ujemanja vzorcev (in stavka `case`)?
  - okolje nas opozori, če pozabimo na primer vzorca
  - okolje nas opozori, če podvojimo vzorec
  - izognemo se okoliščinam, ko na podatkovnem tipu uporabimo napačno metodo za pridobitev vrednosti (npr. `valOf` na vrednosti `NONE` ali `hd` na seznamu `[]`)
  - lažje delo s funkcijami ➔ sledi v nadaljevanju

- kdaj vendarle uporabiti funkcije za preverjanje PT in ekstrakcijo podatkov (`null`, `hd`, `tl`)?
  - v argumentih funkcijskih klicev
  - kadar je preglednost programa večja

Primer: aritmetični izrazi

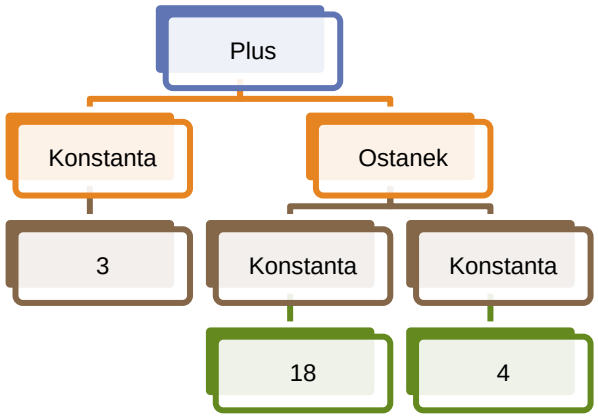
- definirajmo izraz kot **rekurzivni** (!) podatkovni tip

```
datatype izraz =  Konstanta of int
                | Negiraj of izraz
                | Plus of izraz * izraz
                | Minus of izraz * izraz
                | Krat of izraz * izraz
                | Deljeno of izraz * izraz
                | Ostanek of izraz * izraz
```

- primer izraza

```
Plus (Konstanta 3, Ostanek(Konstanta 18, Konstanta 4)
```

- izraze lahko predstavimo z drevesno strukturo

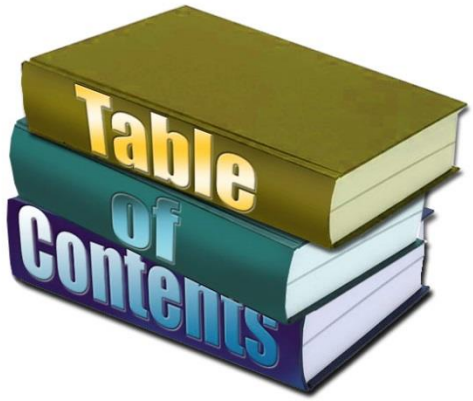


Naloge: aritmetični izrazi

Napiši funkcije tipa `fn : izraz -> int`, s katerimi:

1. evalviraj vrednost aritmetičnega izraza
2. preštej število negacij v izrazu
3. poišči maksimalno konstanto v izrazu
4. poišči število primerov, kjer je ostanek pri deljenju enak 0

Pregled



- sestavljeni podatkovni tipi
  - zapisi (angl. records)
- sinonimi za podatkovne tipe
- izdelava lastnih podatkovnih tipov
- ujemanje vzorcev s stavkom case
- definicija seznama in opcije
- polimorfizem podatkovnih tipov
- ujemanje vzorcev pri deklaracijah

Resnica o seznamih in opcijah

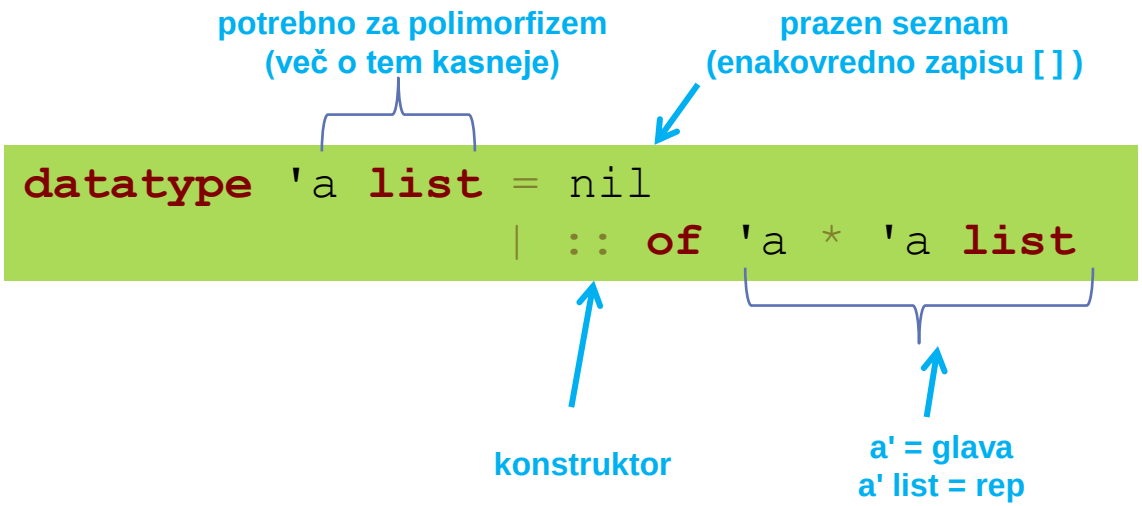
- le sintaktična olepšava v programskem jeziku (niso nujno potrebna komponenta
- definirana sta kot rekurzivna podatkovna tipa
- iz dokumentacije SML:
  - SEZNAM

```
datatype 'a list = nil
                 | :: of 'a * 'a list
```

  - OPCIJA

```
datatype 'a option =  NONE
                   | SOME of 'a
```

Seznami kot rekurzivni podatkovni tip



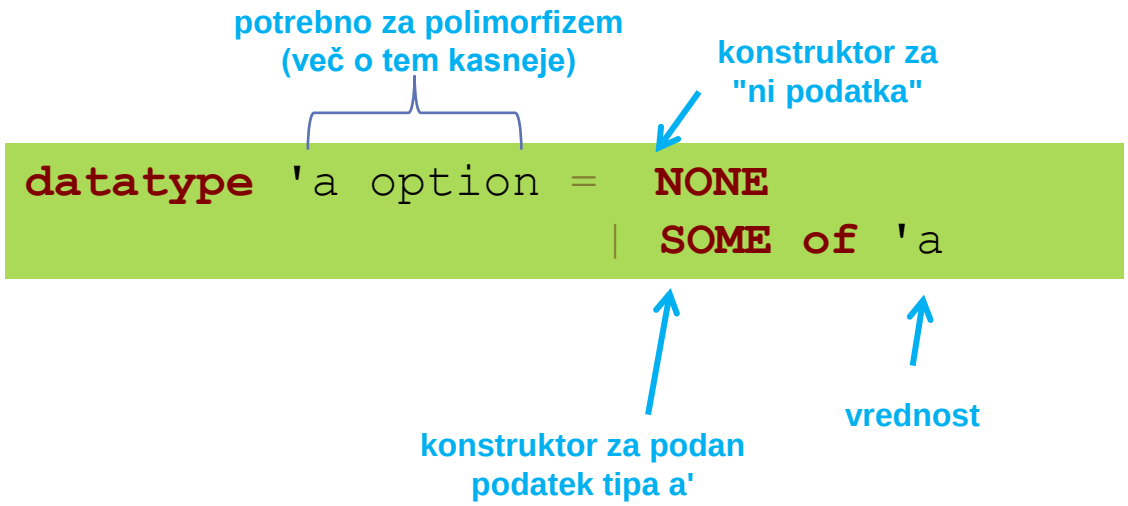
- posebnost: konstruktor `::` je definiran kot infiksni operator (izjema), zato ne moremo zapisati `::(glava, rep)`, temveč pišemo `glava::rep`

```
- 3::5::1::nil;
val it = [3,5,1] : int list
```

- ker seznami uporabljajo konstruktorje, lahko tudi na njih izvajamo ujemanje vzorcev (namesto uporabe `hd`, `tl`, `null`)

- funkcije `hd`, `tl` in `null` znamo sedaj sprogramirati sami!

Opcija kot rekurzivni podatkovni tip



- tudi pri opcijah lahko sedaj uporabimo ujemanje vzorcev
- funkcije `valOf` in `isSome` znamo sedaj sprogramirati sami!

Polimorfizem podatkovnih tipov

- novi podatkovni tip lahko uporablja poljuben drugi (vgnezdeni) podatkovni tip
- zahtevamo konsistentno rabo vgnezdenega tipa (pri vseh pojavitvah predstavlja `'a` isti tip; enako velja za `'b`, `'c` itd.)

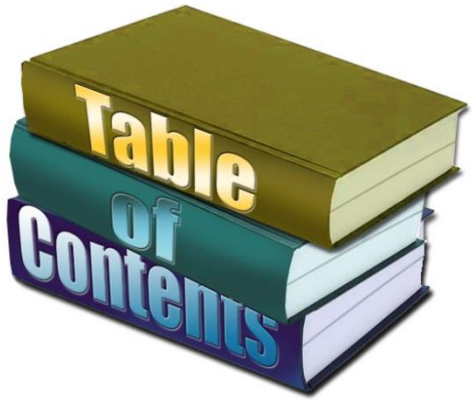
```
datatype 'a list = nil
                 | :: of 'a * 'a list
```

```
datatype 'a option =  NONE
                    | SOME of 'a
```

- primer: izdelajmo lasten polimorfen podatkovni tip: seznam, ki hrani dva različna tipa podatkov:

```
datatype ('a, 'b) seznam =
        Elementa of ('a * ('a, 'b) seznam)
      | Elementb of ('b * ('a, 'b) seznam)
      | konec
```

Pregled



- sestavljeni podatkovni tipi
  - zapisi (angl. records)
- sinonimi za podatkovne tipe
- izdelava lastnih podatkovnih tipov
- ujemanje vzorcev s stavkom case
- definicija seznama in opcije
- polimorfizem podatkovnih tipov
- ujemanje vzorcev pri deklaracijah

Resnica o deklaracijah

- deklaracije spremenljivk in funkcij dejansko uporabljajo **ujemanje vzorcev** na mestu, kjer smo navajali ime spremenljivke:

```
val vzorec = e
fun ime vzorec = e
```

- zgornje pomeni, da vsaka **funkcija sprejema natanko en argument**, ki ga primerja z vzorcem
- ekvivalentna zapisa:

```
fun sestej1 (trojcek: int*int*int) =
    let val (a,b,c) = trojcek
    in a+b+c
    end
```

```
fun sestej2 (a,b,c) =    (* vzorec *)
    a + b + c
```

je kakšna razlika med zapisom z vzorcem in zapisom "funkcije s tremi argumenti"?

Kaj smo se danes naučili?

enostavni podatkovni tipi:
int, bool, real, char, string

sestavljeni podatkovni tipi:
- opcija
- seznam
- terka
- zapis

terka je samo sintaktična olepšava za zapis!

sinonimi za podatkovne tipe (type)
poenostavitev dela

opcij, seznama

definicija opcije in seznama

polimorfizem podatkovnih tipov!
a', b', c' !

lastnih tipov

lastni podatkovni tipi (datatype)
- alternative
- rekurzivna def.

uporaba: stavek case

ujemanje vzorcev

enako pri deklaracijah
```
val p = e
fun ime p = e
```

funkcija sprejme natanko 1 argument
za primerjavo z vzorcem


Repna rekurzija,
funkcije višjega reda