martedì 1 ottobre 2019 11:56

```
let split l =
  let f x y = match y with
              (l1,l2) -> ....
  in foldr f ([],[]) l;;
```

complicato

```
let f x (l1,l2) =
  if x<0 then (x::l1,l2)
         else (l1,x::l2)
```

minmax

```
let rec minmax l =
    match l with
      [x] -> (x, x)
    | x::y::ys -> let (m1, m2) = minmax(y::ys)
                  in
                  if x < m1 then (x, m2)
                  else if x > m2
                       then (m1, x)
                       else (m1, m2);;
```

```
minmax : 'a list -> 'a * 'a = <fun>
```

```
let minmax l =
    let f x (m1, m2) =
        if x < m1 then (x, m2)
        else if x > m2 then (m1, x)
        else (m1, m2)
    in match l with
        x::xs -> foldr f (x, x) xs ;;
```

```
in foldr f (hd l, hd l) (tl l)
```

Date una lista di coppie di interi vogliamo la lista delle somme degli elementi di ogni coppia

sum [(3,5); (6,3); (1;-1); (10,1)] =

[8; 9; 0; 11]

```
let rec sum l =
        match l with
    [] -> []
  | (x,y)::xs -> x+y :: sum xs;;
sum : (int * int) list -> int list = <fun>
```

mercoledì 4 dicembre 2019 12:17

Con foldr

```
# let sum l =
    let f (n,m) y = n+m :: y
    in foldr f [] l ;;
sum : (int * int) list -> int list = <fun>
```

Funzione gli ultimi n elementi di una lista (se ci sono)

conc 3 [1;2;3;4;5] = [1;2]

conc 3 [1;2] = []

```
let rec len l =
        match l with
          [] -> 0
        | x::xs -> 1 + len xs ;;
len : 'a list -> int

let rec conc n l =
        match l with
          [] -> []
        | x::xs -> if len (x::xs) <= n
                   then []
                   else x::conc n xs ;;
```

conc : int → 'a list → 'a list

conc 2 [3;4;5]

3 :: (conc 2 [4;5])

[]

3 :: [] = [3]

```
let rec drop n l =
    match (n, l) with
  (0, l) -> l
| (n, []) when n > 0 -> []
| (n; x::xs) when n > 0 -> drop (n-1) xs;;
```

drop : int -> 'a list -> 'a list = <fun>

```
let conc n l =
    rev (drop n (rev l));;
```

funzione che cancella gli elementi in fondo a una lista e mi dice quanti me ha cancellati

```
let conc n l =
  let rec conc_n n l =
    match l with
      [] -> ([], 0)
    | x::xs -> let (l1, m) = conc_n n xs
               in if m < n
                  then (l1, m+1)
                  else (x::l1, m)
  in let (l1, m) = conc_n n l
     in l1;;
```

m = n

conc_n 3 [1;2;3;4;5] = ([1;2], 3)

conc_n 2 [1;2;3] = ([1], 2)

= let (l1, m) = conc_n 2 [2;3] ([], 2)

$(\square, 2)$

$= \text{let } (l1, m) = \text{conc\_m } 2\ [2;3]$

$l1 = \square$

$m = 2$

in ....

$(1::\square, 2)$

$([1], 2)$

conc_m 2 [2;3]

$(\square, 1)$

$= \text{let } (l1, m) = \text{conc\_m } 2\ [3]$

$l1 = \square$

$m = 1$

in

$(\square, 2)$

conc_m 2 [3]

$=$

$(\square, \emptyset)$

$\text{let } (l1, m) = \text{conc\_m } 2\ \square$

$l1 = \square$

$m = \emptyset$

in

$(\square, 1)$

conc_m 2 []

$= (\square, \emptyset)$

```
# let n = 5 in let m = 6 in n + m
- : int = 11
```

```
let f ...
in let m = f ...
in m ;;
```

Conc n l con foldr

```
let conc n l =
  let f x (l1, m) =
    if m <= n then (l1, m+1)
              else (x :: l1, m)
  in foldr f ([], 0) l ;;
```

dà come risultato una coppia

```
let (l1, m) = foldr f ([], 0) l
  in l1 ;;
```

```
let conc n l =
  let f x (l1, m) =
    if m = 0 then (x :: l1, m)
             else (l1, m - 1)
  in let (l1, m) = foldr f ([], n) l ;;
  in l1 ;;
```

```
match l with
  x::xs
  ↑
```

```
match l with
  l1 @ l2 →
  ═══════   NO
  l1 @ []
  ↑         No!
```

Cancellare tutti gli elementi che
seguono l'ultima occorrenza del
valore ∅. Se ∅ non occorre si
cancellano tutti gli elementi

conc [1;∅;2;3;∅;10;2] = [1;0;2;3;∅]

conc [1;2;3] = []

```
let rec member el l =
    match l with
    [] -> false
  | x::xs -> if x = el then true
             else member el xs;;
member : 'a -> 'a list -> bool = <fun>

let rec conc l =
    match l with
    [] -> []
  | x::xs -> if member ∅ x::xs
             then x::conc xs
             else [];;
```

else {} ;;