giovedì 30 novembre 2017 16:08

una funzione che somma tutti gli elementi di una lista di interi

- ricorsivo
- utilizzando la foldr

Def. ricorsiva

```
let rec sum l = match l with
    [] -> 0
  | x::xs -> x + sum xs ;;
```

sum xs + x ;; ≡

sum : int list → int = <fun>

---

con foldr

foldr f a l

↑ funzione per il fold

f : 'a → 'b → 'b
= =

f è tale che i suoi due argomenti x e y sono rispettivamente:

x è l'elemento di lista che stiamo esaminando

y è il risultato che vogliamo ottenere calcolato su tutti gli elementi che seguono x nella lista.

giovedì 30 novembre 2017 16:20

somma di tutti gli elementi

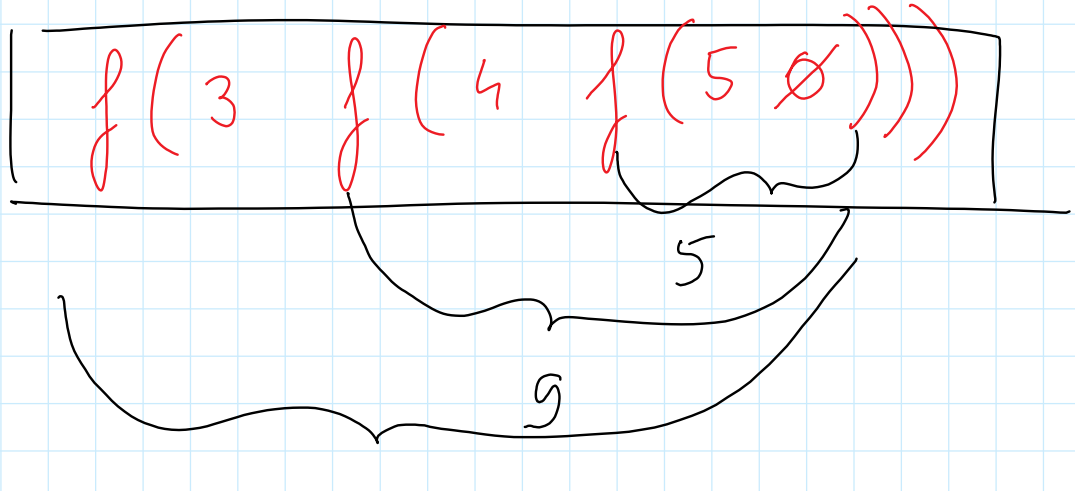
f x y

elemento di lista che stiamo esaminando

somma di tutti gli elementi di seguito x

folder f a l ma la somma di tutti gli elementi di l

```
let f x y = x + y ;;
f : int -> int -> int = <fun>
folder f 0 (3;4;5) ;;
- : int = 12
```



12

Voglio calcolare, mediante l'uso della folder, la coppia (max, min) degli elementi massimo e minimo della lista.

f x y ← risultato nella parte seguente: y è una coppia.

elemento di lista

```
let f x (m,n) = if x > m then (x,n)
                else if x < n
                     then (m,x)
                     else (m,n);;
```

f: 'a → 'a * 'a → 'a * 'a = <fun>

```
let maxmin l = match l with
    x::xs -> folder f (x,x) l;;
```

```
let maxmin l =
    let f x (m,n) = ......
    in match l with
    x:xs -> foldr f (x,x) l ;;
```

xs

Definire la funzione filter con foldr

```
let filter p l =
    let f x y = if p x then x::y
                       else y
    in foldr f [] l ;;
```

funzione map utilizzando foldr

map f [3;4;5] = [f 3; f 4; f 5]

```
let map f l =
  let g x y = f x :: y
  in foldr g [] l ;;
```

significato degli argomenti

Dividere una lista in due liste, la prima con tutti gli elementi <0 e l'altra con tutti gli elementi $>=0$

Risultato è una coppia di liste

```
let rec split l = match l with
    [] -> ([],[])
  | x::xs -> let (l1,l2) = split xs
             in if x<0 then (x::l1, l2)
                else (l1, x::l2);;
```

```
let split l =
  let f x (l1,l2) = if x<0
                    then (x::l1,l2)
                    else (l1,x::l2)
  in foldr f ([],[]) l;;
```

split [-2;-3;4]

$= \{$ def. split, 2° p, $([-3],[4]) = \underline{\text{split}\ (-3;4)}$

$([-2;-3],[4])$

split $[-3,4)$;;

$= \{$ def. 2°, $([],[4]) = \text{split}\ [4]\}$

$([-3],[4])$

split $[4]$

$= \{$ def ... - - 2° p, $([],[]) = \text{split}\ []$

$([],[4])$

```
let split l =
    let f x (l1, l2) = if x < 0
                       then (x::l1, l2)
                       else (l1, x::l2)
    in foldr f ([], []) l ;;
```

split [-2; -3; 4]

f (-2 f (-3 f (4 ([], []))))

([], [4])

([-3], [4])

([-2; -3], [4])