FUNZIONI DI ORDINE SUPERIORE

→ abbiamo iniziato a vedere alcune funzioni di ordine superiore di uso comune

- FORALL
  EXISTS
- MAP

→ Queste funzioni ci consentono di scrivere funzioni che non fanno un uso esplicito della ricorsione

Le funzioni che non richiamano loro stesse ma che usano la ricorsione di MAP, EXISTS, ...

→ ESEMPIO: funzione che raddoppia tutti gli elementi di una lista

```
# let raddoppia l =
    let f x = 2 * x
    in
      map f l ;;
```

NON rec

raddoppia non richiama

ESPLICITAMENTE
se stessa

```
# raddoppia [1;2;3] => [2;4;6]
```

NOTA IL TIPO DEL PARAMETRO DI `f` DEVE ESSERE LO STESSO DI QUELLO DEGLI ELEMENTI DELLA LISTA

ESEMPIO: lista di coppie $[(f_1, x_1); (f_2, x_2); \dots]$

supponiamo di voler calcolare ↓

$[f_1 x_1 ; f_2 x_2 ; \dots]$

Risultato dell'applicazione di $f_i\ x_i$

```
# let f1 x = x+1;;
# let f2 x = x+2;;
# let l = [(f1, 10); (f2, 4); (f1, 8)];;

# let applicaTutti l =
    let apply (f,x) = f x
    in
      map apply l;;

# applicaTutti l ⇒ [11; 6; 9]
```

FILTER

Data una lista e un predicato (funzione che prende un elemento e restituisce True o false) e restituisce la lista costituita dai soli elementi che soddisfano il predicato

$$[3;4;7;8] \xrightarrow{\text{filter pari}} [4;8]$$

```
#let rec filter p l =
    match l with
      [] -> []
    | x::xs -> if p x then
                 x::(filter p xs)
               else
                 filter p xs ;;
```

```
filter : ('a -> bool) -> 'a list -> 'a list
```

ESEMPIO filtrare gli elementi positivi

```
# let solopositivi l =
    let pos x = x >= 0
    in
      filter pos l ;;

# solopositivi [3;-2;5;0;-8] => [3;5;0]
```

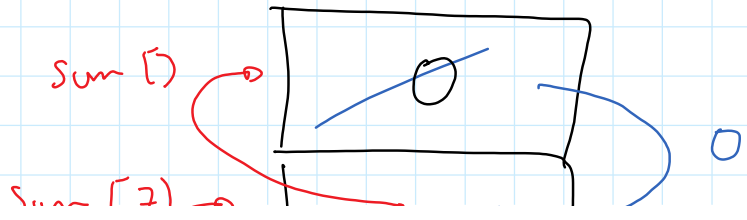
→ Le funzioni che abbiamo visto fino ad ora non mi consentono di lavorare su più di un elemento per volta

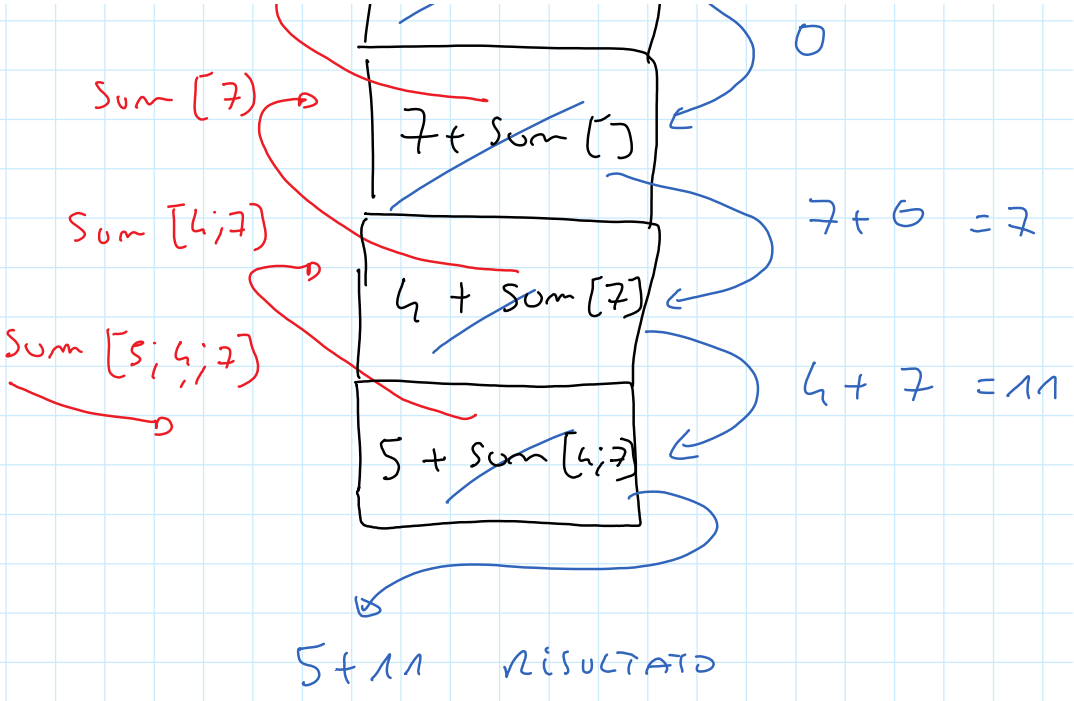
↳ con queste non posso, ad esempio, calcolare la lunghezza o la somma degli elementi di una lista

Ragioniamo su come funziona, ad esempio, la funzione ricorsiva esplicita che calcola la somma

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

Eseguiamo `sum [5;4;7];`

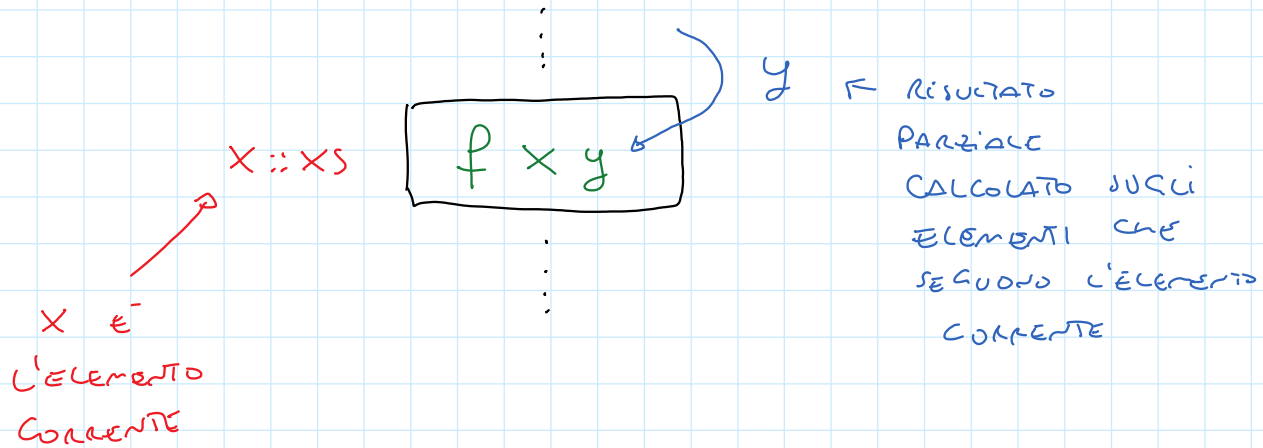




HO COSTRUITO LA PILA DI ATTIVAZIONE FINO A RAGGIUNGERE IL CASO BASE. A QUEL PUNTO

- ABBIAMO IL RISULTATO CALCOLATO SUL CASO BASE []
- SCENDENDO NELLA PILA DI ATTIVAZIONE SI CONSIDERANO GLI ELEMENTI DELLA LISTA UNO ALLA VOLTA E DALL'ULTIMO AL PRIMO
- OGNI ELEMENTO VIENE SOMMATO AL RISULTATO OTTENUTO CALCOLANDO LA SOMMA DEGLI ELEMENTI CHE LO SEGUONO (RISULTATO DELLA CHIAMATA RICORSIVA PRECEDENTE - VIENE DA SOPRA NELLA PILA)

DESCRIVIAMO QUESTO COMPORTAMENTO CON UNA FUNZIONE DI ORDINE SUPERIORE



FOLDR

A DIFFERENZA DEI CASI PRECEDENTI f PREVEDE 2 PARAMETRI x e y

```
# let rec foldr f a e =
    match e with
    [] -> a
  | x::xs -> f x (foldr f a xs)
```

RISULTATO DEL CASO BASE []

calcola y

```
foldr : ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b
```

('a -> 'b -> 'b) : f ; 'b : a ; 'a list : f ; 'b : risultato

USIAMO FOLDR PER CALCOLARE LA SOMMA DEGLI ELEMENTI DI UNA LISTA

```
# let sum l =
    let f x y = x + y
  in
    foldr f 0 l ;;
```

ELEMENTO CORRENTE

SOMMA DEI SUCCESSIVI

CASO []

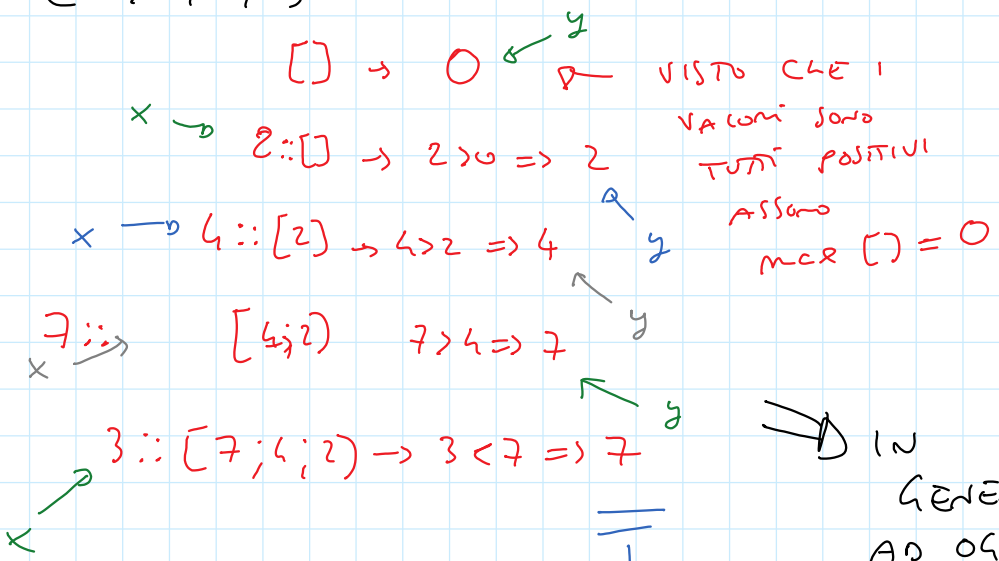
ESEMPIO

CALCOLARE LA LUNGHEZZA

```
# let len l =
    let f x y = 1 + y
  in
    foldr f 0 l ;;
```

ESEMPIO : calcolare il valore massimo in una lista (ASSUMIAMO CHE I VALORI NELLA LISTA SIANO TUTTI POSITIVI)

[3;7;4;2) => 7



QUALI SONO QUINDI a ed f DA PASSARE A fold?

a 0

f x y if (x>y) Then x else y

QUINDI:

let max l =

```
let f x y = if x > y then x else y
in
foldr f 0 e
```

ESEMPIO massimo di una lista che può assumere valori anche negativi

(la funzione non è definita su liste non vuote)

↳ PROBLEMI: - il caso base [] non è definito
- da quale massimo iniziale partiamo?

IDEA: ESTRAGGO IL PRIMO VALORE E LO USO COME MASSIMO NEL CASO []

[7; 4; 8; 2]

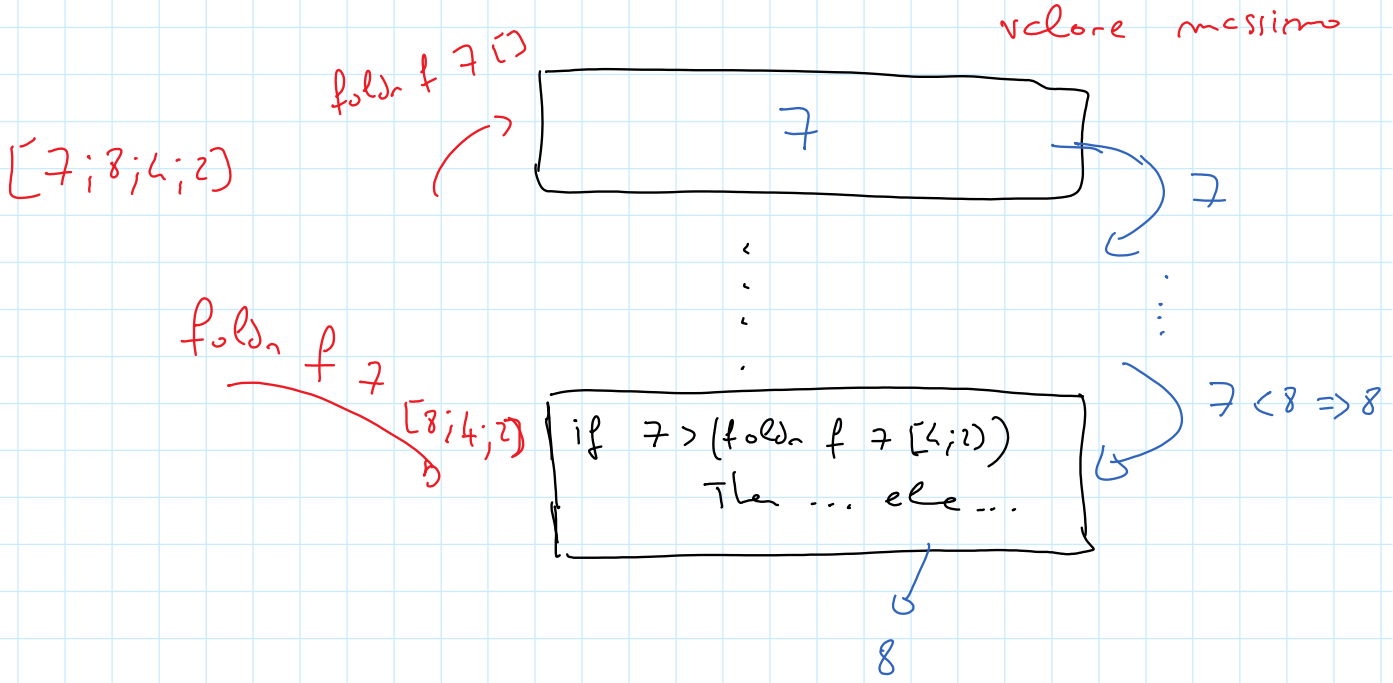
↳ USO LA fold SU (4; 8; 2) USANDO 7 COME a

7

```
# let max l =
    let f x y = if x > y then x else y
    in
    match l with
```

```
x::xs -> foldn f x xs ;;
```

foldn calcola su xs (che può essere vuoto) usando il primo elemento x come primo possibile valore massimo



[5;0;4;2;0;9) = 8+5

venerdì 1 dicembre 2017 10:53

ESEMPIO: CALCOLARE LA SOMMA DEGLI ELEMENTI PRECEDONO IL PRIMO ZERO NELLA LISTA. SE NON CI SONO ZERI FA LA SOMMA DI TUTTO

[8;5;0;4;2;0;9] ⇒ 13 (8+5)

[] = 0

9::[] = 9+0 = 9 x+y

0::[9] = 0

2::[0;9) = 2+0 = 2 x+y

4::[2;0;9) = 4+2 = 6 x+y

0::[4;2;0;9) = 0

5::[0;4;2;0;9) = 5+0 = 5

8::[5;0;4;2;0;9] = 8+5 = 13

se x=0 RIPARTE LA SOMMA ...

```
# let sommaprima0 e =
    let f x y = if x=0 then 0
                else x+y
    in
```

foldn f o e ;;