FUNZIONI DI ORDINE SUPERIORE

- map : $('a \to 'b) \to 'a\ list \to 'b\ list$

  $map\ f\ [x_1; \ldots; x_n] = [f\ x_1; \ldots; f\ x_n]$

- filter : $('a \to bool) \to 'a\ list \to 'a\ list$

  che elimina da una lista tutti gli elementi che non soddisfano il predicato primo argomento

- foldr : operatore di fold

foldr f a $[x_1; \ldots ; x_n]$ =

$$f\ x_1\ (f\ x_2\ (\ldots\ (f\ x_n\ a)\ldots))$$

è il risultato di foldr quando la lista è vuota
(elemento neutro)

f è una funzione BINARIA (a due argomenti)

CALCOLARE LA SOMMA DEGLI ELEMENTI DI UNA LISTA

$$\sum_{i=1}^{n} x_i$$

$$[x_1; \ldots; x_n]$$

$$x_1 + \sum_{i=2}^{n} x_i$$

risultato parziale del calcolo sulla lista $[x_2; \ldots; x_n]$

↳ si aggiunge al risultato parziale il contributo di x_1

$$+ \; x_1 \; (+ \; x_2 \; \ldots \; (+ \; x_n \; \emptyset) \ldots)$$

$$\prod_{i=1}^{n} x_i = * \; x_1 \; (* \; x_2 \; \ldots \; (* \; x_n \; 1) \ldots)$$

o ——— o

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

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

('a -> 'b -> 'b) : f

'b : a

'a list : l

'b : res

$\sum_{i=1}^{n} x_i$ mediante foldr

```
# let sum x y = x + y ;;
sum : int -> int -> int
```

```
let sum = prefix + ;;
```

```
# let sumlist l = foldr sum 0 l ;;
sumlist : int list -> int
```

l res.

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

sumlist [2; -1; 4]

= { def. di sumlist }

foldr sum ∅ [2; -1; 4]

= { 2° patt. di foldr }

sum 2 (foldr sum ∅ [-1; 4])

= { 2° patt. di foldr }

sum 2 (sum -1 (foldr sum ∅ [4]))

= { 2° patt. di foldr }

sum 2 (sum -1 (sum 4 (foldr sum ∅ [])))

= { 1° patt. di foldr }

sum 2 (sum -1 (sum 4 ∅))

```
let rec foldr f a l =
    match l with
    [] -> a
    x::xs -> f x (foldr f a xs);;
```

= {def di sum}

2 + (-1 + (4 + 0))

= 5

```
# let prod x y = x * y ;;
  prod : int -> int -> int

# let prodlist l = foldr prod 1 l ;;
  prodlist : int list -> int
```

o ——— o

```
# let forall p l = let f x y = (p x) & y in
                   foldr f true l
```

forall p [x₁; ... ; xₙ] =

f x₁ (f x₂ (... (f xₙ true) ...))

= (p x₁) & (p x₂) & ... & (p xₙ) & true

```
let exists p l = let f x y = (p x) or y
                 in
                 foldr f false l ;;
```

$$\underbrace{(p\ x_1) \vee (p\ x_2) \vee \ldots \ldots \vee (p\ x_n)}$$

COME SI RISOLVE UN PROBLEMA UTILIZZANDO FOLDR

```
let foo l = let f x y = ....
            in foldr f ? l
```

x → un generico elemento di l

y → risultato di foo se applicata alla lista che segue x in l

in modo che foo l calcoli il numero di elementi maggiori di 0 di una lista di interi

---

$$foo\ [x_1; \dots; x_n] = foldr\ f\ ?\ [x_1; \dots; x_n]$$

$$= f\ x_1\ (\dots\ (f\ x_i\ (f\ x_{i+1}\ \dots\ (f\ x_n\ ?)\dots)))$$

$$foo\ [x_{i+1}; \dots; x_n] = foldr\ f\ ?\ [x_{i+1}; \dots; x_n]$$

```
let foo l = let f x y = if x > 0
                        then 1 + y
                        else y
            in
               foldr f 0 l
```

↳ numero di elementi maggiori di 0 nella porzione di l che segue x

f x y deve aggiungere il contributo di x al risultato

foo [-1; 2]

= foldr f ∅ [-1; 2]

= f (-1) (foldr f ∅ [2])

= f (-1) (f 2 (foldr f ∅ []))

= { 1° patt. di foldr }

f (-1) (f 2 ∅)

= f (-1) 1

= 1

f x y = if x > 0 then
1 + y
else y

if 2 > 0 then 1 + ∅ else ∅

if (-1 > 0) then 1 + 1 else 1

CALCOLARE IL NUMERO DI ELEMENTI e LA LORO SOMMA IN UNA LISTA

$$g\ [x_1; \ldots; x_n] = \left(n, \sum_{i=1}^{n} x_i\right)$$

```
let rec g l = match l with
    []    ->  (0, 0)
  | x::xs ->  let (m, s) = g xs in
              (m+1, s+x) ;;
```

se calcolo g xs = (m, s)

```
let g l = let f x (m,s) = (m+1, s+x)
                                 15+1
          in foldr f (0,0) l
```

```
g [1;10;5]
= foldr f (0,0) [1;10;5]
= {2° p. di foldr}
  f 1 (foldr f (0,0) [10;5])
= {2° p. di foldr}
  f 1 (f 10 (foldr f (0,0) [5]))
= {2° p. di foldr}
  f 1 (f 10 (f 5 (foldr f (0,0) [])))
= {1° p. di foldr}
  f 1 (f 10 (f 5 (0,0)))
= f 1 (f 10 (1,5))
= f 1 (2,15)
= (3,16)
```

Se non avessimo utilizzato il p. m. per rappresentare il risultato della chiamata ricorsiva nella prima def. di g, ovvero il secondo argomento di f nella def. di g mediante folder

```
let rec g l = match l with
    [] -> (0,0)
  | x::xs -> let (m,s) = g xs
             in (m+1, s+x)
```

```
let first (x,y) = x ;;
let second (x,y) = y ;;
```

```
x::xs -> (1 + first (g xs), x + second (g xs))
```

Domanda: voglio definire f utilizzando g, h, come funzioni locali.

```
let f .... = let g .... = ....
             in
                let h ..... = .....
             in
```

```
let f ..... = let g .... = ....
              and
                  h ..... = .....
              in
                  ......
```

Portare, se c'è, l'ultimo elemento di una lista in testa alla lista.

foo [10; 5; 6] = [6; 10; 5]

let foo l = let f x y =

se y è della forma
w::ws ⟶ w::x::ws

[.... x; [.. xs ..]]

y è xs in cui l'ultimo elemento è stato portato in testa

in

.... foldr ....

```
let foo l = let f x y = match y with
                [] -> [x]
              | w::ws -> w::x::ws
            in foldr f [] l ;;
```

foo [10;20;30]

= foldr f [] [10;20;30]

= f 10 (foldr f [] [20;30])

= f 10 (f 20 (foldr f [] [30]))

= f 10 (f 20 (f 30 (foldr f [] [])))

= f 10 (f 20 (f 30 []))

= f 10 (f 20 [30])

= f 10 (30::20::[]) =

= 30::10::20::[]

MAP e FILTER attraverso fold.

```
let rec map f l = match l with
    []      ->  []
  | x::xs   ->  (f x) :: (map f xs) ;;
```

```
let mapp f l = let g x y = (f x)::y
```

in

```
fold g [] l
```

g: generico elemento di l

y: risultato parziale così map f xs dove xs è tutto ciò che segue x in l

```
let rec filter p l = match l with
    []      ->  []
  | x::xs when (p x)  ->  x::(filter p xs)
  | x::xs when not (p x)  ->  filter p xs ;;
```

```
let filter p l = let f x y = if (p x) then x::y
                             else y
in
fold f [] l
```