Funzioni di ORDINE SUPERIORE (al primo)

Titolo nota 24/09/2015

Funzioni che hanno come ARGOMENTO o danno come RISULTATO altre funzioni

---

```
# let apply f x = f x ;;
apply : ('a -> 'b) -> 'a -> 'b = <fun>
```

argomento funzionale

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

```
# sum 5 ;;
- : int -> int = <fun>
```

il risultato è una funzione

Funzioni di ordine superiore "UTILI"

Titolo nota 24/09/2015

Quantificatori su liste

- Quantificatore universale: controlla che un predicato sia vero su tutti gli elementi di una lista

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

tipo predicato — tipo lista — tipo risultato

funzione che ha come risultato un valore di verità (bool)

```
let pari x = x mod 2 = 0 ;;
pari : int -> bool = <fun>
```

tipo x — tipo ris

```
forall pari [2;3;4];;
- : bool = false

forall pari [2;4;12];;
- : bool = true
```

forall

Titolo nota 24/09/2015

```
# let rec forall p l = match l with
      []      ->  true
    | x::xs   ->
        if p x then forall p xs
               else false ;;
```

```
forall : ('a -> bool) -> 'a list -> bool = <fun>
```

('a -> bool): tipo p; 'a list: tipo l; bool: tipo xs

```
[]      -> true
| x::xs when p x     -> forall p xs
| x::xs when not(p x) -> false ;;
```

Quantificatore esistenziale

Titolo nota 24/09/2015

exists : vale vero se e solo se ESISTE un elemento di una lista su cui è vero un predicato

```
exists pari [2;3;7;10];;
- : bool = true

exists pari [];;
- : bool = false

exists pari [3;5];;
- : bool = false
```

---

exists : ('a -> bool) -> 'a list -> bool = <fun>

exists

Titolo nota 24/09/2015

```
# let rec exists p l = match l with
      [] -> false
    | x::xs ->
        if p x then true
               else exists p xs
```

```
| [] -> false
| x::xs when p x -> true
| x::xs when not(p x) -> exists p xs;;
```

map

Titolo nota 24/09/2015

Mappa una funzione su tutti gli elementi di una lista

```
map : ('a -> 'b) -> 'a list -> 'b list = <fun>
```

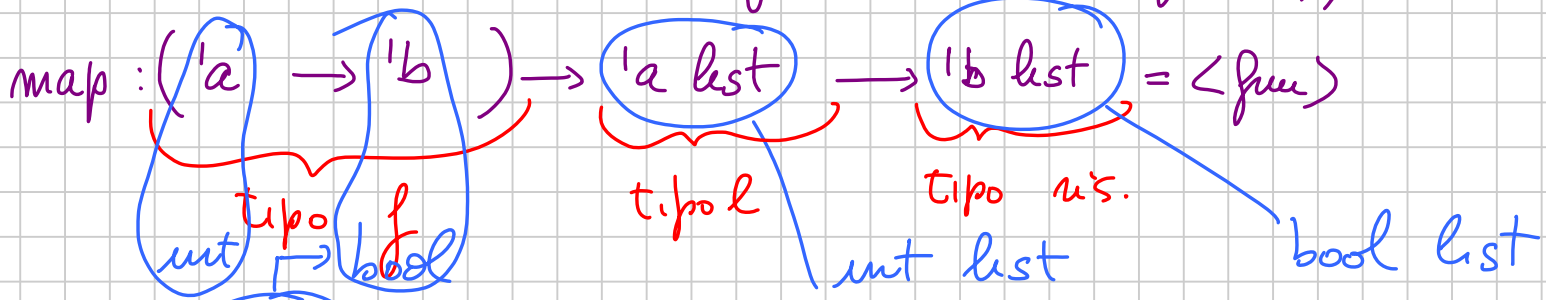
```
# let inc x = x+1;;
inc : int -> int = <fun>

# map inc [1;2;3];;
- : int list = [2;3;4]
```

```
# map pari [3;4;5];;
- : bool list = [false; true; false]
```

map

Titolo nota 24/09/2015

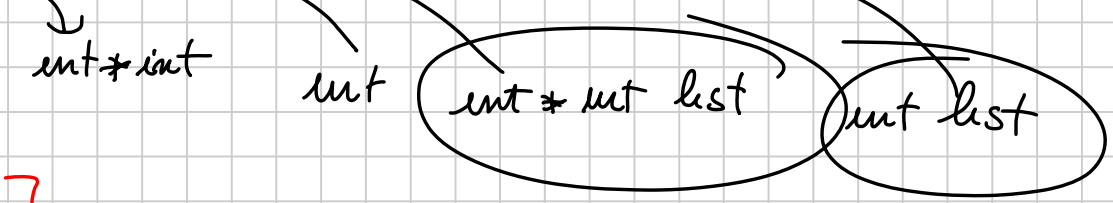
```
# let rec map f l = match l with
                    [] -> []
                  | x::xs -> f x :: map f xs ;;
```




```
# map pari [1;2;3];;
- : bool list = [false; true; false]
```

Titolo nota 24/09/2015

map : ('a → 'b) → 'a list → 'b list




```
let sum (x,y) = x+y ;;
sum : int * int -> int = <fun>

map sum [(3,2);(4,3);(-1,-1)] ;;
- : int list = [5;7;-2]
```

```
let apply (f,x) = f x ;;
apply : ('a -> 'b) * 'a -> 'b = <fun>
```

tipo di (f,x)

tipo uscita

Titolo nota 24/09/2015

map : ('c → 'd) → 'c list → 'd list

let incr1 x = x+1 ;;

incr1 : int → int = <fun>

let incr2 x = x+2 ;;

incr2 : int → int = <fun>

let apply (f, x) = f x ;;

apply : ('a → 'b) * 'a → 'b = <fun>

'c = ('a → 'b) * 'a

'd = 'b

map apply [(incr1, 1); (incr2, 3)] ;;

('a → 'b) * 'a

int → int * int

('a → 'b) * 'a

int → int * int

- : int list = [2; 5]

filter

Titolo nota 24/09/2015

filter : ('a -> bool) -> 'a list -> 'a list = <fun>

filter mantiene nel risultato tutti gli elementi della lista che soddisfano il predicato

es:

```
# filter pari [1;2;3;4;5];;
- : int list = [2;4]
```

filter

Titolo nota 24/09/2015

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

```
[ ] -> [ ]
| x::xs when p x -> x :: filter p xs
| x::xs when not(p x) -> filter p xs;;
```

filter : ('a -> bool) -> 'a list -> 'a list = <fun>

('a -> bool): tipo p

'a list: tipo l

'a list: tipo risultato

Titolo nota 24/09/2015

filter : ('a → bool) → 'a list → 'a list

'a = int

filter pari [2;3;4];;

int → bool

int list int list

- : int list = [2;4]

---

let sum10 (x,y) = x+y = 10 ;;

sum10 : int * int → bool = <fun>

filter sum10 [(3,5);(6,4);(4,4);(5,5)];;

- : int* int list = [(6,4);(5,5)]

filter : ('a → 'b) → 'a list → 'a list

int* int list

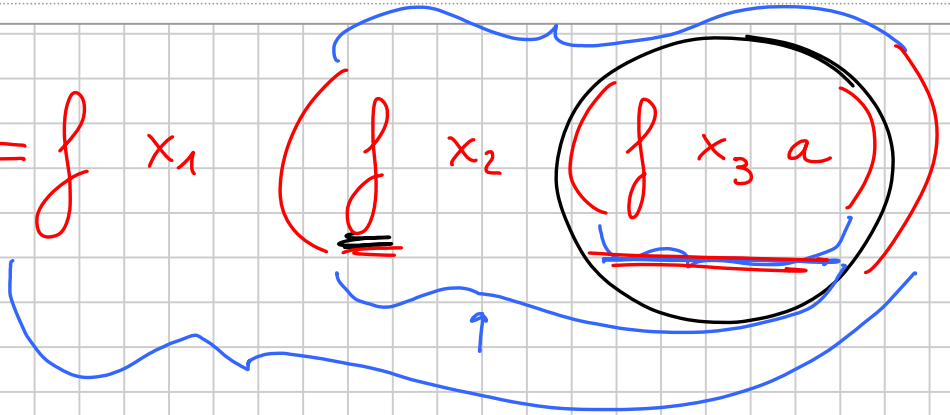
'a : int*int

'b : bool

Titolo nota 24/09/2015

foldr

foldr f a [x1 ; x2 ; x3] = f x1 (f x2 (f x3 a))



nella f che relazione c'è tra il tipo del secondo argomento e quello del risultato (della f)?

f : 'a → 'b → 'b

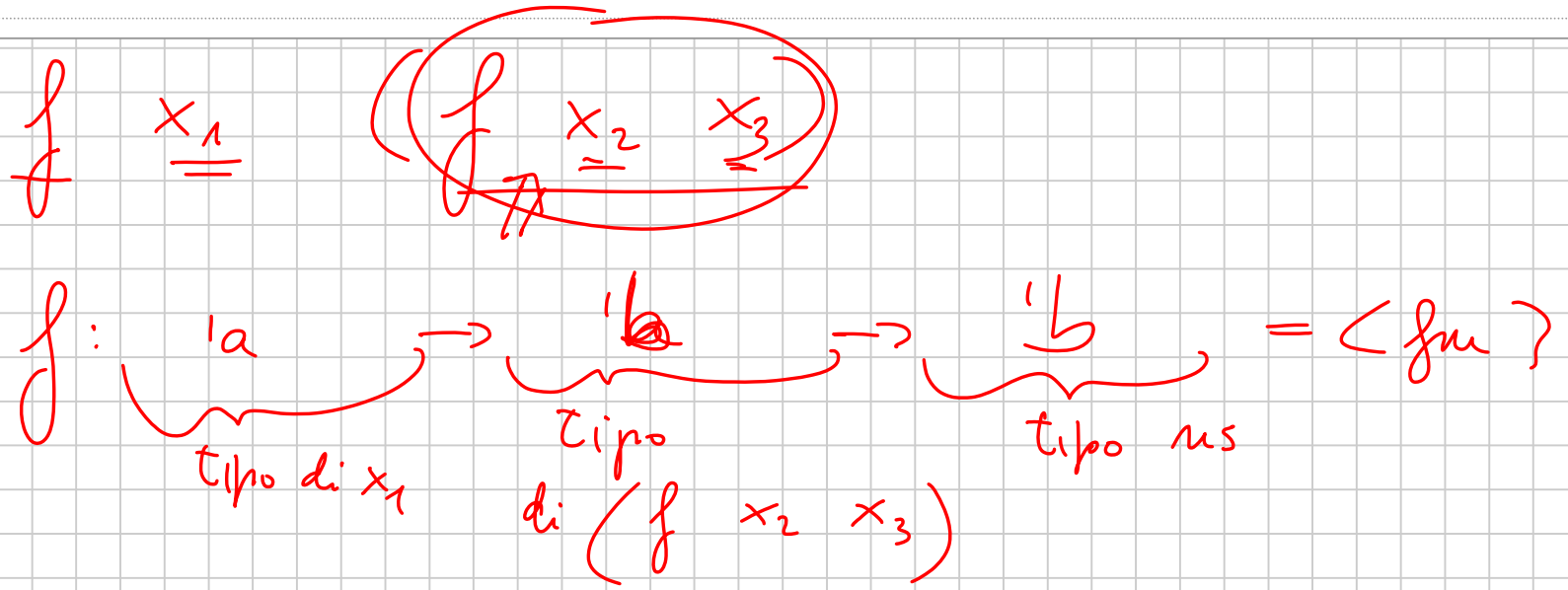
=

f : int → bool → int

f 5 (f 3 true)

int

Titolo nota 24/09/2015



Titolo nota 24/09/2015

$$foldr\ f\ a\ [x_1; x_2; x_3] = f\ x_1\ (f\ x_2\ (f\ x_3\ a))$$

```
# 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 = <fun>
```

tipo f — tipo a — tipo l — tipo ris.

Titolo nota 24/09/2015

```
# let g x y = y+1;;
g : 'a -> int -> int = <fun>
```

```
  foldr g ∅ [2;3;4]
= { def foldr, 2° p }
  g 2 (foldr g ∅ [3;4])
= { def foldr, 2° p }
  g 2 ( g 3 (foldr g ∅ [4])) = { def g }
= { " " }
  g 2 ( g 3 ( g 4 ( foldr g ∅ [])))
```

```
= { def foldr, 1° p }
  g 2 ( g 3 ( g 4 ∅ ))
= { def g }
  g 2 ( g 3 1 )
= { def g }
  g 2 2
= { def f }
  3
```

Risultato di foldr g ∅ [2;3;4] è la lunghezza della lista

Titolo nota 24/09/2015

```
let s x y = x + y ;;
```

x: elemento che stiamo analizzando attualmente

y: risultato su tutti gli elementi seguenti nella lista

```
fold_r s ∅ [1;2;3]
= {def fold_r, 2° p}
  s 1 (fold_r s ∅ [2;3])
= { " }
  s 1 (s 2 (fold_r s ∅ [3]))
= { " }
  s 1 (s 2 (s 3 (fold_r s ∅ [])))
= {def fold_r, 1° p}
  s 1 (s 2 (s 3 ∅))
= {def s}
  s 1 (s 2 3)
= {def s}
  s 1 5
= { " }
  6
```

3: risultato su tutti gli elementi seguenti nella lista

us. risultati elementi seguenti

Titolo nota 24/09/2015

filter mediante foldr

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

```
filter pari [2;3;4];;
= {def filter}
foldr f [] [2;3;4]
= {def foldr 2° p}
f 2 (foldr f [] [3;4])
= {"}
f 2 (f 3 (foldr f [] [4]))
= {"}
f 2 (f 3 (f 4 (foldr f [] [])))
= {def foldr, 1° p}
f 2 (f 3 (f 4 []))
```

```
f 2 (f 3 [4])
= {def f}
f 2 [4] = 2::[4] = [2;4]
```