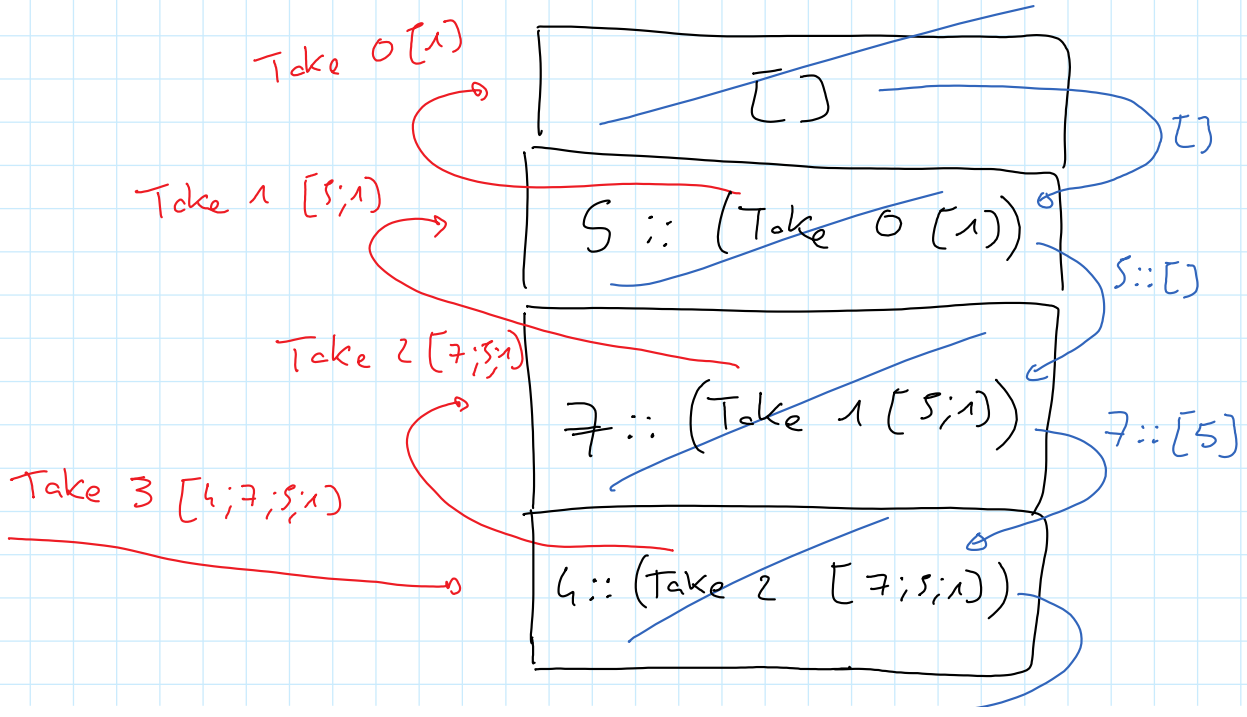
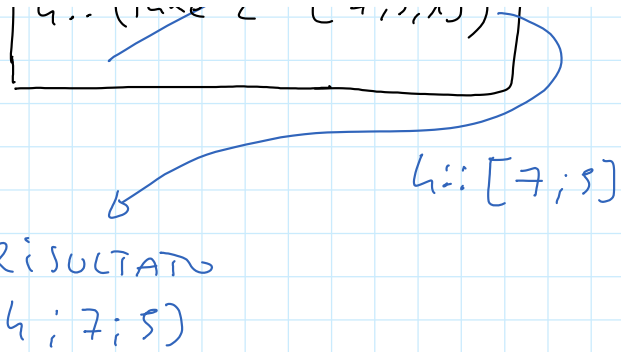
- ESEMPIO: funzione che prende una lista e un valore n e restituisce i primi n elementi della lista

Take 2 [4;7;8;2] => [4;7]
Take 0 [4;7;8;2] => []
Take 2 [] => []

```
# let rec Take n l =
    match (n, l) with
      (0, l1) -> []
    | (n1, []) -> []
    | (n1, x::xs) -> x :: (Take (n1-1) xs);;
```

ESEGUIAMO Take 3 [4;7;5;1];;




4:: [7;5]
RISULTATO
(4;7;5)

Che cosa succede se chiamo

Take -3 [7;4;2;1];; ?

ENTRA SEMPRE NELL'ULTIMO CASO DEL MATCH RICHIAMANDOSI RICORSIVAMENTE CON VALORI SEMPRE PIÙ BASSI -4 -5 -6 ... METTENDO UNO DOPO L'ALTRO TUTTI GLI ELEMENTI DELLA LISTA NEL RISULTATO.

TERMINA RAGGIUNGENDO IL CASO BASE

(n1, [])

ossia Take -3 [7;5;4;1] => [7;5;4;1]

MODIFICHIAMO Take IN MODO CHE DIA [] SE n < 0

```
# let rec Take n l =
    match (n, l) with
      (n1, l1) when n1 <=0 -> []
    | (n1, []) -> []
    | (n1, x::xs) -> x::(Take (n-1) xs);;
```

≠ (n1, l1) -> if (n1<=0) Then []
else ....

modifichiamo Take in modo che non sia definita per $n < 0$

```
#let rec Take n l =
    match (n, l) with
      (0, l1) -> []
    | (n1, []) when n1>0 -> []
    | (n1, x::xs) when n1>0 ->
        x::(Take (n1-1) xs);;
```

PATTERN MATCHING NON ESAUSTIVO!

es. (-5, [3;7;2]) non fa match con nessun pattern.

Take: int -> 'a list -> 'a list

ESEMPIO funzione che prende una lista e restituisce True se è ordinata in modo non decrescente (ogni elemento è ≥ del precedente). false altrimenti

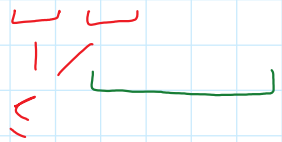
[3;5;8;8;12]

DEVO CONFRONTARE OGNI COPPIA DI ELEMENTI UNO SUCCESSIVO ALL'ALTRO

CASI BASE [] True

[7] True

[3;7;14;28]



LA LISTA È ORDINATA SE IL 1° ELEMENTO È <= DEL 2° E LA LISTA CHE VA DAL 2° ELEMENTO IN POI È ORDINATA!!

```
#let rec ord l =
    match l with
      [] -> True
    | x::[] -> True
    | x::xs -> if (x<= hd xs)
                 Then ord xs
                 else false ;;
```

else false ;;

POSSO SCRIVERLO ANCHE COSI'

```
| x::y::xs -> if x<=y Then and(y::xs)
                else false
```

FA MATCH CON LISTE DI ALMENO 2 ELEMENTI

PER ESERCIZIO

→ definire una funzione drop che data una lista e un numero n restituisce la lista di tutti gli elementi escluso i primi n

drop 2 [7; 8; 4; 5] => [4; 5]

→ definire una funzione ennesimo che restituisce l'elemento in posizione n nella lista

→ definire una funzione sommacostante che prende una lista di coppie e verifica che la somma degli elementi di ogni coppia sia sempre uguale per tutti gli elementi della lista

sommacostante [(1,5); (3,3); (0;6)]

↘ True

Provate a risolverlo senza usare

fst e snd

PUNTO DELLA SITUAZIONE

Abbiamo visto esempi di funzioni ricorsive su liste

idea
1) Trovo i casi base
2) penso a come risolvere i casi non base utilizzando chiamate ricorsive su liste più piccole (es. senza il primo elemento)

QUESTO MODO DI DEF. FUNZIONI RICORSIVE SI DICE

"CON RICORSIONE ESPLICITA"

LA FUNZIONE RICHIAMA SE STESSA

FUNZIONI DI ORDINE SUPERIORE (AL PRIMO)

Sono funzioni che prevedono un parametro che è un'altra funzione

→ abbiamo già visto

```
# let apply f x = f x ;;
```

PRENDE UNA FUNZIONE — E LA APPLICA

Vediamo che cosa ci consente di fare:

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

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

POTREMMO METTERE A FATTORE COMUNE

TUTTE LE PARTI UGUALI DI QUESTE DUE

FUNZIONI

FORALL verifica una condizione su TUTTI gli elementi di una LISTA

PREDICATO: funzione che prende un elemento e restituisce True o false

```
# let rec forall p l =
    match l with
      [] -> True
    | x::xs -> if not p x Then false
               else forall p xs;;
```

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

ESEMPIO

```
# let TuttiPari l =
    let pari x = (x mod 2 = 0)
    in
      forall pari l;;
```

NON È RICORSIVA

```
# let TuttiPositivi l =
    let pos x = x > 0
    in
      forall pos l;;
```

EXISTS

verifica se c'è un elemento nella lista che soddisfa un predicato p

SIMILE A FORALL

X ESERCIZIO

MAP

Map applica una funzione ricevuta come argomento a tutti gli elementi della lista

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

ESempio incrementa di 1 tutti gli elementi

```
# let incrementa l =
    let f x = x+1
    in
    map f l ;;
```

incrementa [1;2;3] => [2;3;4]