

# FUNZIONI AUSILIARIE LOCALI

# Per  $f_x =$

$$\text{Per } g(z) = z * z$$

in

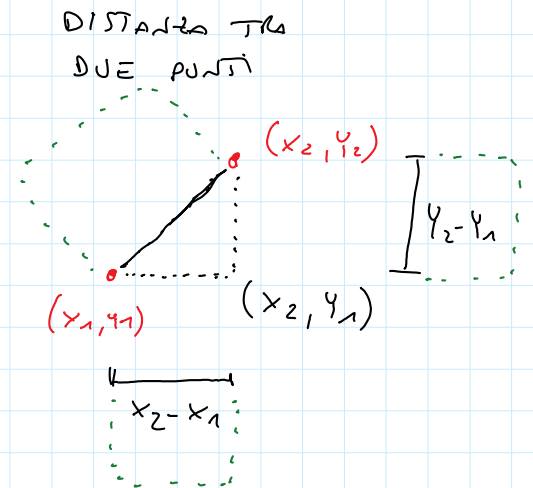
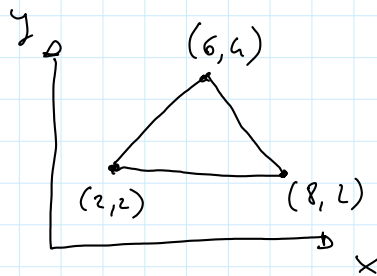
$$g(x) + 1; j$$

$$f(x) = g(x) + 1$$

$$\text{dove } g(z) = z * z$$

ESEMPIO PIU' COMPLESSO

→ Scriviamo una funzione che calcola il perimetro di un Triangolo date le coordinate dei Tre angoli:



DISTANZA TRA DUE PUNTI (TEOREMA DI PITAGORA)

$$\text{dist}((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

# let perimetro p1 p2 p3 =

let dist (x1, y1) (x2, y2) =

$$\text{sqrt} \left( (x_2 - x_1) * (x_2 - x_1) + (y_2 - y_1) * (y_2 - y_1) \right)$$

funzione  
predefinita  
√

in

dist p1 p2 +.

dist p2 p3 +.

dist p3 p1 ;;

perimetro : float \* float → float \* float → float \* float → float

↑                      ↑                      ↑                      ↑  
P1                      P2                      P3                      risultato

- IL TIPO DI P1 È ... \* ... PERCHÉ VIENE PASSATO A DIST CHE SI ASPETTA UNA COPPIA
- È UNA COPPIA DI float PERCHÉ IN DIST FACCIO +, \*. SUI SUOI ELEMENTI
- IL RISULTATO HA TIPO FLOAT (SOMMA +. DEI RISULTATI OTTENUTI DA DIST)

```
# perimetro (0.0, 0.0) (3.0, 0.0) (0.0, 4.0);  
- : float = 12.0
```

# ALTO ESEMPIO

→ Scriviamo una funzione che prende un parametro

float x e bool y

Se y è True calcola l'area di un cerchio di raggio x. Se y è false calcola l'area di un quadrato di lato x

```

# let f x y =
  let quadrato z = z * z
  in
    if y then
      let pi = 3.14
      in quadrato x * pi
    else
      quadrato x ;;

```

VISIBILITÀ  
di  
QUADRATO

VISIBILITÀ  
di  
pi

VISIBILITÀ  
di  
z

POTREI  
USARE x e y  
ANCHE QUI  
VISIBILITÀ  
di x e y

f : float → bool → float

# f 10.0 True;;

- : float = 314.0

# f 10.0 false;;

- : float = 100.0

# DEFINIZIONI RICORSIVE

venerdì 24 novembre 2017 09:41

$$\text{fact}(m) = \begin{cases} 1 & \text{se } m=0 \\ m * \text{fact}(m-1) & \text{altrimenti} \end{cases} \quad \text{fact}: \mathbb{N} \rightarrow \mathbb{N}$$

## let rec fact m = if m=0 then 1  
else m \* fact (m-1);;

DICE A  
CAML CHE  
LA FUNZIONE  
È RICORSIVA

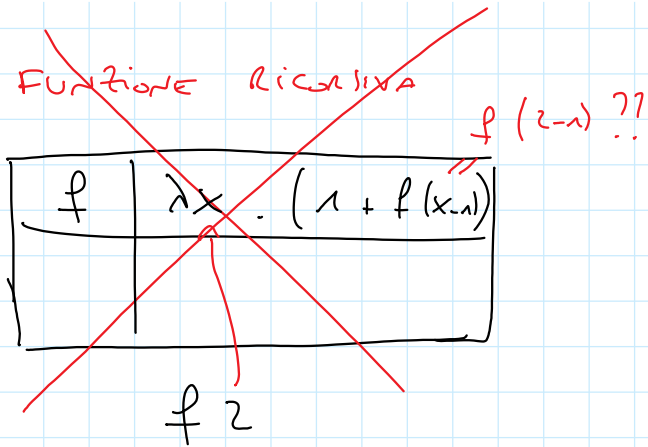
↑ ?  
ci vogliono  
( )  
ALTRIMENTI APPLICA  
fact A m E  
POI SOTTRA 1

BISOGNA SCRIVERE let rec PERCHÉ CAML  
DEVE CALCOLARE QUESTA FUNZIONE DI VARSAMENTE  
RISPETTO AL SOLITO

FUN NON RICORSIVA

f	$\lambda x. (x+1) = 3$
⋮	
⋮	

f 2

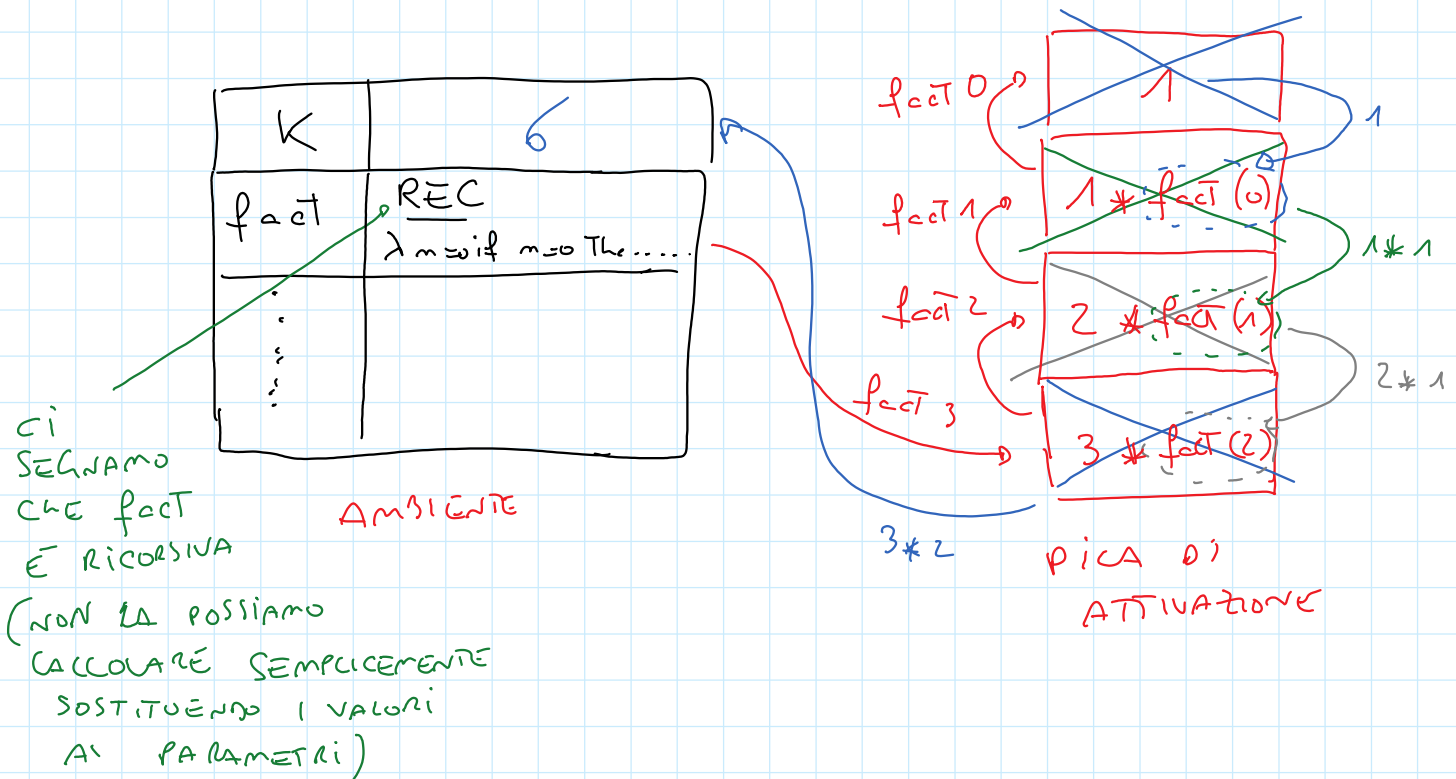


PER DESCRIVERE CHE  
COSA SUCCEDERE QUANDO  
SI APPLICA UNA FUNZIONE  
RICORSIVA USIAMO UNA  
PILA DI ATTIVAZIONE

# ESECUZIONE DI FUNZIONI RICORSIVE: PILA DI ATTIVAZIONE

venerdì 24 novembre 2017 09:47

```
#let k = fact 3;;
```



## ESEMPIO FUNZIONE DI FIBONACCI

$$\text{fibonacci}(n) = \begin{cases} 0 & \text{se } n=0 \\ 1 & \text{se } n=1 \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & n > 1 \end{cases}$$

```
#let rec fibo n =
```

```
  if n=0 Then 0
```

```
  else if n=1 Then 1
```

```
  else fibo (n-1) + fibo (n-2);;
```

## PER ESERCIZIO

- DEFINIRE RICORSIVAMENTE  
funzione  $\text{pow } x \ y$  che calcola  $x^y$   
- (elevamento a potenze)
- funzione  $\text{modulo } x \ m$  che calcola  
 $x \text{ modulo } m$  ( $x \% m$  in C)



## LISTE IN CAML

→ Una lista è una sequenza omogenea di valori  
 Tutti dello stesso tipo

→ La lista in Caml è un tipo primitivo!!  
 (come gli int, float, coppie, ...)

```
# [7; 4; 5];;
```

```
- : int list = [7; 4; 5]
```

TIPO  
"LISTA di int"

```
# [3.4; 7.0; 8.2; 4.1];;
```

```
- : float list = [3.4; 7.0; 8.2; 4.1]
```

```
# [True; false; True];;
```

```
- : bool list = [True; false; True]
```

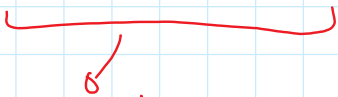
posso fare liste di qualunque tipo, anche liste di funzioni!!

```
# let f x = x + 1;;
```

# Let  $g(x) = x - 1$ ;

#  $[f; g]$ ;

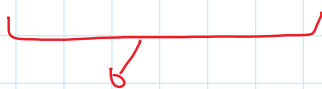
$f : \text{int} \rightarrow \text{int}$   $\text{List} = [\langle f \rangle, \langle f \rangle]$

  
Lista di  
funzioni  $\text{int} \rightarrow \text{int}$

Anche Liste di Liste:

#  $[[3; 4]; [0; 1; 2]]$

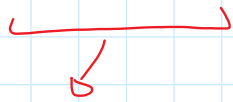
- :  $\text{int}$   $\text{List}$   $\text{List} = [[3; 4]; [0; 1; 2]]$

  
Lista di Liste di  $\text{int}$

LA LISTA VUOTA si RAPPRESENTA  $[\ ]$

$\# [\ ] ; ;$

- : 'a list = (fun)



lista di elementi di qualunque tipo 'a

## OPERAZIONI SULLE LISTE

-> hd (head) restituisce il primo elemento di una lista

$\# \text{hd } [6; 5; 4] ; ;$

- : int = 6

-> tl (Tail) restituisce la lista ottenuta rimuovendo il primo elemento

$\# \text{tl } [6; 5; 4] ; ;$

- : int list = [5; 4]

NOTA : hd e tl NON SONO DEFINITE SU LISTE VUOTE !!

# hd [],;;  
ERRORE!

#tl L);;  
ERRORE!!

→ :: (cons) aggiunge un elemento in  
Testa ad una lista

*ELEMENTO*  
↑  
# 4 :: [7;8];;  
*LISTA*  
↑  
- : int list = [4;7;8]

|| IL TIPO DEL  
NUOVO ELEMENTO  
DEVE ESSERE LO  
STESSO DI QUELLO  
DELLA LISTA

# 4 :: [];;  
- : int list = [4]

← in TESTA A []  
POSSO AGGIUNGERE  
UN ELEMENTO DI  
QUALUNQUE TIPO  
OTTENENDO UNA LISTA  
DI QUEL TIPO

→ @ (append) concatena due liste  
dello stesso tipo

# [3;4]@[7;8];;  
- : int list = [3;4;7;8]

## FUNZIONI RICORSIVE SU LISTE

→ CALCOLO DELLA LUNGHEZZA DI UNA LISTA

idea: la lunghezza di  $[8; 7; 4; 2]$   
 è pari a  $1 +$  la lunghezza  
 di  $[7; 4; 2]$ , che a  
 sua volta è  $1 +$  la lunghezza  
 di  $[4; 2]$  che è  
 $= 1 + \text{lunghezza}[2]$   
 $\hookrightarrow = 1 + \text{lunghezza}[]$

↳ QUESTA È  
 UGUALE A 0 !!

# let rec len l =

if l=[] then 0

else  $1 + \text{len } (tl\ e)$ ;;

↳  
 se l non è vuota  
 la sua lunghezza è  
 $1 +$  la lunghezza di  $tl\ e$   
 $\downarrow$   
 TAIL

→ SOMMA DEGLI ELEMENTI DI UNA LISTA  
 IDEA: e' data dal primo elemento + la  
 somma di tutti i successivi

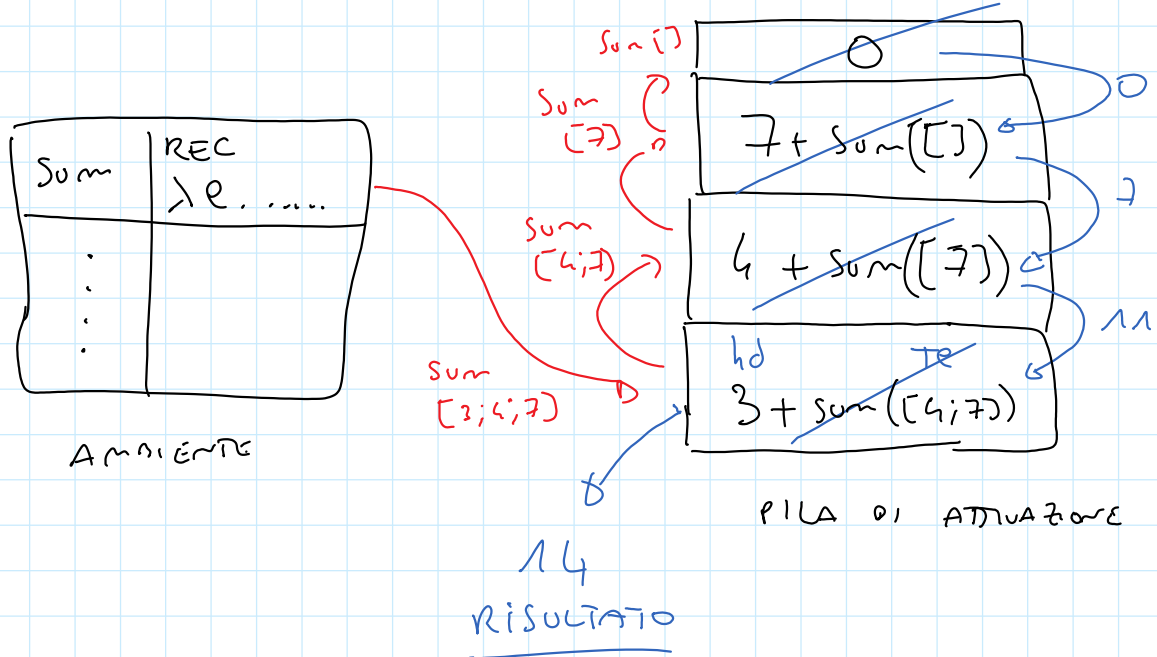
let rec sum l =

if l = [] then 0

else hd l + sum (tl e)

PRIMO ELEMENTO  
 chiamata ricorsiva sul resto della lista

ESECUZIONE di sum [3;4;7];



## PATTERN MATCHING (match)

match è un costrutto che consente di confrontare il risultato di un'espressione con una sequenza di casi possibili

match ESPRESSIONE with

```

    PATTERN1 → ESPRESSIONE1
  | PATTERN2 → ESPRESSIONE2
    ⋮
  | PATTERNm → ESPRESSIONEm
  
```

ESEMPIO: NEGAZIONE di UN BOOLEANO

```
# let negazione b =
```

```
    match b with
```

```
      True  → false
```

```
    | false → True ;;
```

```
negazione: bool → bool = <fun>
```

```
# negazione (2 < 5) ;;
```

```
- : bool = false
```

Un pattern (modello) può essere:

NOTA:  
UNA COSTANTE!!  
PUÒ PUÒ ESSERE  
UNA FUNZIONE →

- un valore : True , false , 0 , 1 , 2 , 3.4 , 7.2 , ...
- una variabile : x , y , z , ...
- una coppia (o enupla) di pattern :

(3,7) (x,y) (x,7,5,y)

- una lista vuota : []
- una lista di pattern ottenuta utilizzando cons :

$x :: []$       $x :: e$   
 $(x,y) :: (z,k) :: e$



Un valore "fa match" (soddisfa) un pattern se esiste un modo di istanziare le variabili contenute nel pattern che consenta di ottenere il valore con cui lo stiamo confrontando

match V with  
 $\vdots$   
 $P \rightarrow \dots$

<u>PATTERN P</u>	<u>VALORI V</u>	
True	True	✓ FA MATCH
True	false	✗ NON FA MATCH
3	3	✓
3	4	✗
(3,4)	(3,4)	✓
(3,4)	(4,3)	✗
(x,4)	(8,4)	✓ x=8 y=4
(x,7)	(8,7)	✓ x=8
(x,7)	(3,2)	✗ IN QUALUNQUE MODO ISTANZIATO X NON OTTENGO (3,2)
[]	[]	✓
8::[]	[8]	✓
8::[]	[9]	✗
8::[]	[7,5]	✗

[ ]	[ ]	✓
8 :: [ ]	[ 8 ]	✓
8 :: [ ]	[ 9 ]	✗
8 :: [ ]	[ 7 ; 5 ]	✗

x :: [ ]	[ 7 ]	✓	x = 7
x :: [ ]	[ True ]	✓	x = True
x :: [ ]	[ ]	✗	
x :: [ ]	[ 2 ; 3 ]	✗	

[ x ; y ] ≡ x :: y :: [ ]	[ 2 ; 3 ]	✓	x = 2 y = 3
x :: y :: [ ]	[ 1 ]	✗	
x :: e	[ ]	✗	
x :: e	[ 7 ]	✓	x = 7 e = [ ]
x :: e	[ 2 ; 4 ; 8 ]	✓	x = 2 e = [ 4 ; 8 ]

ATTENZIONE  
NON POSSO VIARE  
@ (append)  
NEI PATTERN

SE POTESSE SCRIVERE

e1 @ e2

CI SAREBBERO  
PIU' MODI DIVERSI  
PER MATCHARE  
INIZI [ 1 ; 2 ; 3 ; 4 ]

E' UN  
PROBLEMA!

PIU' MODI  
PER MATCHARE  
IL VALORE [1;2;3;4]

$$e1 = [1;2]$$

$$e2 = [3;4]$$

MA ANCHE

$$e1 = [1]$$

$$e2 = [2;3;4]$$

E' UN  
PROBLEMA...

QUINDI  
NO @ NEI PATTERN

# Rivediamo $len$ usando PATTERN MATCHING

# let rec  $len$   $l$  =

match  $l$  with

$[]$  → 0

|  $x :: xs$  → 1 +  $len$   $xs$  ;;

LE VARIABILI DEL PATTERN POSSONO ESSERE USATE NELLA CORRISPONDENTE ESPRESSIONE

I CASI  
VENGONO  
VALUTATI  
UNO DOPO  
L'ALTRO  
IN SEQUENZA

istanziata  
con il  
1° elemento

istanziata  
con il  
resto della lista

corrisponde  
a  
 $len$  (Tl e)