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

# let succ = sum 1 ;;
succ : int -> int = <fun>
```

succ y = 1 + y

```
# succ 5;;
- : int = 6

# let inc3 = sum 3;;
inc3 : int -> int = <fun>

# inc3 5;;
- : int = 8
```

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

```
# sum 3;;
- : int -> int = <fun>
```

la funzione sum è in grado (quando applicata a un solo argomento) di restituire una funzione come risultato

Se le funzioni possono dare come risultato altre funzioni si dicono

di ORDINE SUPERIORE

(al primo)

Se le funzioni possono avere come risultato funzioni, possono avere anche funzioni come argomento?

```
# let apply f x = (f (x+1))+1;;
```

$$\text{apply} : \underbrace{(\text{int} \to \text{int})}_{\text{tipo } f} \to \underbrace{\text{int}}_{\text{tipo } x}$$

$$\to \underbrace{\text{int}}_{\text{tipo del ris}} = \text{<fun>}$$

$$\text{apply} : (\text{int} \to \text{int}) \to \text{int} \to \text{int}$$

```
let apply f x = (f (x+1)) + 1;;
apply: (int -> int) -> int -> int = <fun>
let inc2 = sum 2;;
inc2 : int -> int = <fun>

apply inc2 3;;
- : int = 7
```

$$(\text{inc2}\ (3+1)) + 1$$

let mag0 x = x > 0;;

mag0 : $\underbrace{\text{int}}_{\text{tipo x.}} \longrightarrow \underbrace{\text{bool}}_{\text{tipo ris}}$ = <fun>

apply mag0 3;;

errore di tipo;;

apply : (int → int) → int → int

Definizioni ricorsive

```
let rec potenza(x, y) =
    if y = 0 then 1
    else x * potenza(x, y-1);;
```

$$\text{potenza} : \underbrace{\text{int} * \text{int}}_{\text{tipo }(x,y)} \rightarrow \underbrace{\text{int}}_{\text{tipo ris}} = \langle\text{fun}\rangle$$

$$\text{potenza}(10, -1)$$

$$= 10 * \text{potenza}(10, -2)$$

$$= 10 * 10 * \text{potenza}(10, -3)$$

$$\vdots$$

$$\left(\forall n, m \in \mathbb{N} \quad \text{potenza}(n, m) = n^m\right)$$

Precedenze indotte dalla def. di potenza

$$\begin{array}{c|c}
(x, y) & (x, -1) \\
\vert & \vert \\
(x, y-1) & (x, -2) \\
\vdots & \vdots \\
(x, 0) \leftarrow &
\end{array}$$

Caso base

$$potenza(x, 0) = x^{0}$$

Caso induttivo:

$$\left(\forall n, m \in \mathbb{N} \quad potenza(n, m) = n^{m} \implies potenza(n, m+1) = n^{(m+1)}\right)$$

$$potenza(n, 0) = n^0$$

$$potenza(n, 0)$$

$$= \{\text{def potenza, 1° caso}\}$$

$$1$$

$$= \{\text{calcolo}\}$$

$$n^0$$

$$\text{potenza}(n, m) = n^m \xrightarrow{\text{ip. ind.}} \text{potenza}(n, m+1) = n^{(m+1)}$$

$\text{potenza}(n, m+1)$

$= \{\text{def. potenza},\ m+1 \in \mathbb{N},\ m+1 > 0,\ 2^\circ \text{ caso}\}$

$n * \text{potenza}(n, m)$

$= \{\text{ip. induttiva}\}$

$n * n^m$

$= \{\text{calcolo}\}$

$n^{(m+1)}$

```
let rec potenza (n,m) =
  if m=0 then 1
     else n*potenza(n,m-1);;
```

MATCH

PATTERN

```
let rec potenza (n,m) =
     match m with
        0 -> 1
  |     x -> n*potenza(n,m-1);;
```

```
let rec
  potenze (n,m) = match m with
    0 -> 1
  | x when x>0
    -> n*potenze(n,m-1);;

potenze : int * int -> int = <fun>
```

$$\text{let apply } \underbrace{f\ x}_{\text{f e x sono parametri di apply}} = \underbrace{f\ x}_{\text{f è una funzione e viene applicata a x}} ;;$$

let apply f x = f x ;;

$$\text{apply} : \underbrace{('a \rightarrow 'b)}_{f} \rightarrow \underbrace{'a}_{x} \rightarrow \underbrace{'b}_{ris}$$

= <fun>

'a 'b 'c ....

polimorfi

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

let mego x = x>0;;
mego : int -> bool = <fun>

let inc1 x = x+1;;
inc1 : int -> int = <fun>
```

```
apply mego 3;;
- : bool = true
```

```
apply inc1 3;;
- : int = 4
```

```
apply mego true;;
```

ERRORE DI TIPO

```
let rec fact n =
  if n=0 then 1
    else n * fact (n-1) ;;
```

---

se $n<0$ l'esec. è infinita

---

l'applicazione di funzione
ha precedenza
sulle applicazioni
degli operatori

```
let rec fact n = match n with
    0 -> 1
  | x when x>0 -> n * fact(n-1);;
```

$$\text{fact} : \underbrace{\text{int}}_{\text{tipo } n} \to \underbrace{\text{int}}_{\text{tipo ris}} = \langle \text{fun} \rangle$$

```
fact (-3);;
```

indefinito

let rec fact n = match n with

$\emptyset \longrightarrow 1$

| x when x > 0 → n * fact(n-1);;

| x when x > 0 → x * fact(x-1);;

Si!

| n when n > 0 → n * fact(n-1);;

Si!

```
let f x y =
  if x>0 then
    if y>0 then 1
          else 2
  else if y>0 then 3
          else 4;;
```

```
let f x y = match (x, y) with
  (n, m) when n>0 & m>0 -> 1
| (n, m) when n>0 & m≤0 -> 2
| (n, m)  "   n≤0 & m>0 -> 3
| (n, m)  "   n≤0 & m≤0 -> 4 ;;
```

```
let f x y =
  if (x>0 & y>0) then 1
    else if (x>0 & y≤0) then 2
       else - - - .
```

mcd

```
let rec mcd (x,y) =
  if x=y then y
  else if x<y then mcd(x,y-x)
       else mcd(x-y,y);;

mcd : int * int -> int = fun
mcd (21,14);;  - : int = 7
```

```
let rec mcd x y =
  if x = y then x
    else if x < y
      then mcd x (y-x)
      else mcd (x-y) y ;;
```

mcd : int -> int -> int = <fun>