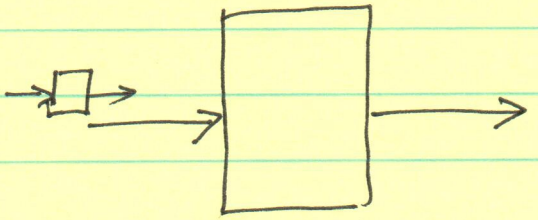
Higher order functions



Higher-order functions specific to lists
I want to apply a given function to every element of a list.

(Built-in) `map : (α→β) → α-list → β-list`

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

`module List` `open List`

`List.map` does not require `open List`.

`mymap inc [1;2;3] ~> [2;3;4]`

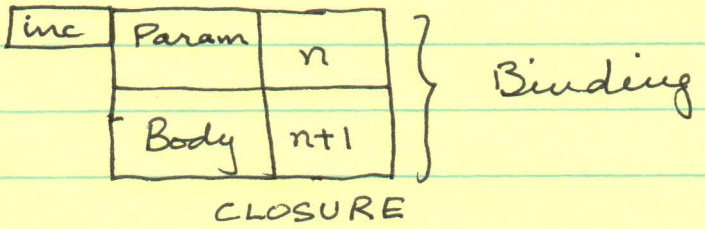
`test : α→bool`

```
let rec filter test l =
  match l with
  | [] -> []
  | h::t -> if (test h) then h::(filter test t)
            else (filter test t)
```

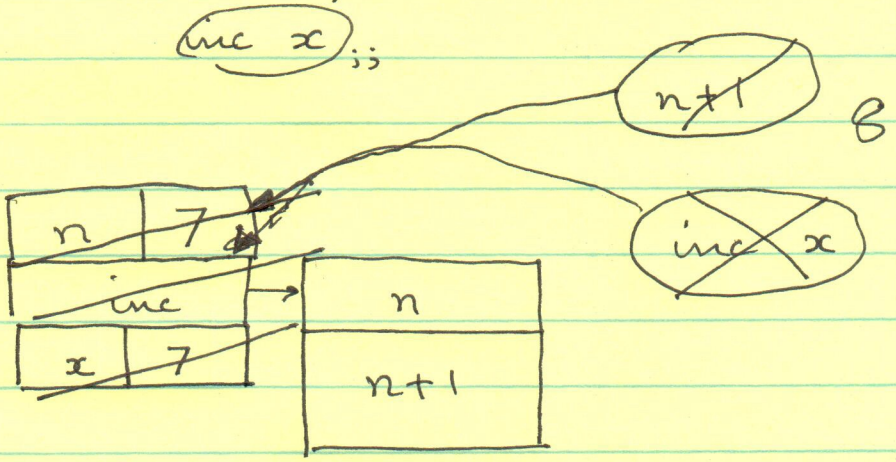
```
init n f
init 5 (fun n→n*n) ~> [0;1;4;9;16]
```

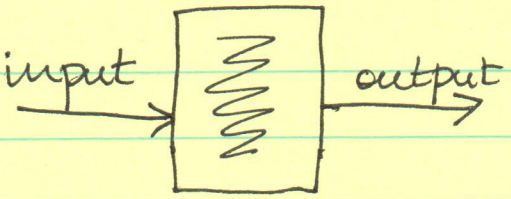
ENVIRONMENT MODEL OF FUNCTIONAL EXPRESSION EVALUATION

```
let inc = fun n -> n+1
```




```
let x = 7 in
  let inc = fun n -> n+1 in
    inc x;;
```





let inc n = n+1

OR

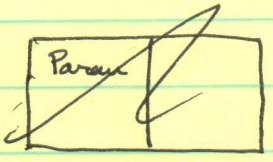
let inc = fun n → n+1

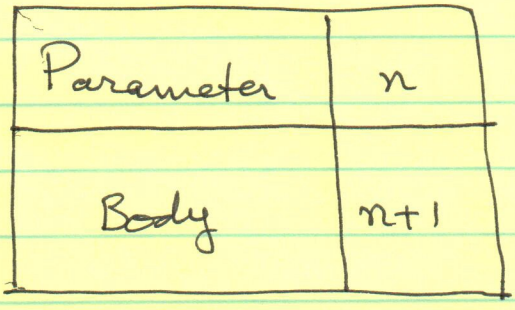
let id = fun x → x IDENTITY

inc is a function

(inc 7) is NOT a function

↳ integer valued expression





SUBSTITUTION : replace parameter by evaluated argument in the body.

(n+1) ⇝ (7+1) ⇝ 8

sin π/2 → 1.0

The most powerful list processing function in the galaxy.

$$fold\_left : (\alpha \to \beta \to \alpha) \to \alpha \to \beta\text{-}list \to \alpha$$

$$fold\_left\ f\ a\ [x_1; x_2; \cdots x_k] = \qquad a : \alpha,\ x_i : \beta$$

$$(f \ \cdots \ (f\ (f\ a\ x_1)\ x_2) \cdots \ x_k)$$

($f : \alpha \to \beta \to \alpha$, $acc : \alpha$, $l : \beta\text{-}list$)

tail recursive

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

```
let sum_list l = fold_left (+) 0 l
let myconcat l = fold_left (@) [] l
```

↳ $\alpha\text{-}list\text{-}list \to \alpha\text{-}list$

```
let stringmash strlst =
  fold_left (fun s1 -> fun s2 -> s1 ^ s2) "" strlst
```

("" : empty)

$O(n)$

```
let rev l = fold_left (fun a x -> x::a) [] l
```

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

let newmap f l =
    fold_right (fun x a -> (f x) :: a) l []
```

NOT tail recursive

fold-left: $(f \cdots (f\ (f\ a\ x_1)\ x_2) \cdots\ x_k)$

fold-right: $f\ \underset{\alpha \to \beta \to \beta}{l}\ \underset{\beta}{b} : \beta$

l : $[x_1; x_2; \cdots x_k]$ (α-list)

$(f\ x_1 \cdots (f\ x_{k-1}\ (f\ x_k\ b)) \cdots)$

type of fold_right

$(\alpha \to \beta \to \beta) \to \alpha\text{-list} \to \beta \to \beta$

Why pure functional programming?

No side-effect e.g. update
→ No control flow
limited sense of computational precedence

We can execute computations in parallel.
Dataflow architecture
↳ computation is triggered by arrival of data