val x = 1;
fun addX y = x+y;
val x = 2;
addX 0; (* = 1 *)

fun listify x = [x]; (* 'a → 'a list *)
listify "hi"; (* = ["hi"], not [0] *)

(* CURRYING *)

fun someCurriedFunction x y = 0;
(* 'a → 'b → int *)
someCurriedFunction 1 "hi";
(* int *)

(* HIGHER-ORDER FUNCTIONS *)

null; (* 'a list → bool *)
map; (* (a → b) → 'a list → 'b list *)
val null_mask = map null; (* 'a list list → bool list *)
val xss = [[], [1,2,3]];
map null xss; (* 'a list list *)

(* Three indistinguishable alternatives: *)
(* 1: discouraged; use builtin map *)
fun negate_list xs = case xs of
| [] ⇒ []
| x::xs' ⇒ ~x :: negate_list xs';
(* 2 *)
fun negate_list xs = map (λ x ⇒ ~x) xs;
(* 3 *)
val negate_list = map (λ x ⇒ ~x);

(* TAIL RECURSION *)

(* A "tail call" is, conceptually a function call inside another function, where
* the caller just returns the callee’s return value without doing any more work.
* *)

(* An expression is "in tail position" iff any:
* it is the ENTIRE right-hand-side of a function body
* it is the ENTIRE then-branch or else-branch of a conditional that’s in tail position
* it is the ENTIRE body of a let-expression in tail position
* it is the ENTIRE right-hand-side of a case expression in tail position
* ...I think that’s it. *)

(* e.g. *)
datatype json = Object of (string * int) list | OmittedJson Constructors;
fun one_fields j = let
  fun loop (fs,acc) = case fs of
    | [] ⇒ acc
    | ([], []) ⇒ acc
    | (_, []) ⇒ loop (fs,k::acc)
  in
    case j of
    | Object fs ⇒ loop (fs,[])
    | _ ⇒ []
  end
end

(* DATATYPES *)

(* each-of *)
type person = (string × int);