exception ListLengthMismatch

(* a list * 'b list * 'c list -> ('a * 'b * 'c) list
('a * 'b * 'c) list -> 'a list * 'b list * 'c list *)

(* don't do this *)
fun old_zip3 (l1,l2,l3) =

(* 'a list * 'b list * 'c list −> ('a * 'b * 'c) list ('a * 'b * 'c) list −> 'a list * 'b list * 'c list * don't do this *)
fun multsign (x1, x2) =

if null l1 ∧ null l2 ∧ null l3
then raise ListLengthMismatch
else (hd1, hd2, hd3) :: old_zip3(tl1, tl2, tl3)

(* don't do this *)
fun shallow_zip3 (l1,l2,l3) =

fun len xs =

fun zip3 list_triple =

exception MyUndesirableCondition

exception MyOtherException of int × int

fun mydiv (x,y) =

fun maxlist (xs,ex) =

fun nondecreasing xs =

fun cumulative_sum xs =

(* nested pattern-matching often convenient even without recursion; also the wildcard pattern is good style
fun fact2 n = 
  let fun aux(n,acc) = if n=0 then acc else aux(n-1,acc*n) 
  in aux(n,1) 
  end 

fun sum1 xs = 
  case xs of [] ⇒ 0 | i::xs' ⇒ i + sum1 xs'

fun sum2 xs = 
  let fun f (xs,acc) = case xs of [] ⇒ acc | i::xs' ⇒ f(xs',i+acc) 
  in f(xs,0) 
  end

fun rev1 xs = 
  case xs of [] ⇒ [] | x::xs' ⇒ (rev1 xs') @ [x]

fun rev2 xs = 
  let fun aux(xs,acc) = case xs of [] ⇒ acc | x::xs' ⇒ aux(xs', x::acc) 
  in aux(xs,[]) 
  end

(* more examples of making functions tail-recursive *)