fun double x = 2 * x

fun triple_n_times3 (n,x) = n_times((let fun triple y = 3 * y), n, x)

(* This does not work: a function /binding/ is not an /expression/ *)

fun triple_n_times4 (n,x) = n_times((λ y ⇒ 3 * y), n, x)

(* This /anonymous function/ expression works and is the best style: *)

val eighteen = (#1 a_tuple) 9

fun triple_n_times5 = λ (n,x) ⇒ n_times((λ y ⇒ 3 * y), n, x)

(* Because triple_n_times4 does not call itself, we could use a val-binding to define it, but the fun binding above is better style *)

val rev_poor = List.rev

fun nth_tail_lame (n,xs) = n_times((λ y ⇒ 1 + increment_n_times_lame(n−1,x)), n, xs)

(* Good style: *)

fun nth_tail_good (n,x) = n_times(tl, n, x)

(* Considering the results so far, it seems that the (λ y ⇒ 3 * y) should always be in there. *)

fun triple_n_times2 (n,x) = let fun triple x = 3 * x in n_times(triple,n) end

(* actually since used only once, we could define it right where we need it *)

fun triple_n_times (n,x) = n_times((λ y ⇒ 3 * y), n, x)

(* This does not work: a function /binding/ is not an /expression/ *)

(* This /anonymous function/ expression works and is the best style: *)

val triple_n_times5 = λ (n,x) ⇒ n_times((λ y ⇒ 3 * y), n, x)

(* Notice the function has no name *)

val glacier = double_or_triple (λ x ⇒ x−3 − 4)

val nine = (double_or_triple (λ x ⇒ x = 42)) 3

(* Higher-order functions over our own datatype bindings *)
datatype exp = Constant of int
    | Negate of exp
    | Add of exp × exp
    | Multiply of exp × exp

fun true_of_all_constants(f,e) =
  case e of
    Constant i ⇒ f i
    | Negate e1 ⇒ true_of_all_constants(f,e1)
    | Add(e1,e2) ⇒ true_of_all_constants(f,e1) ∧ true_of_all_constants(f,e2)
    | Multiply(e1,e2) ⇒ true_of_all_constants(f,e1) ∧ true_of_all_constants(f,e2)

fun all_even_exp e = true_of_all_constants((λ x ⇒ x mod 2 = 0),e)