fun silly1 (z : int) = 
  let
  val x = if z > 0 then z else 34
  in
  if x > y then x else y
  end

fun silly2 () = let
  val x = 1
  in
  (let val x = 2 in x+1 end) + (let val y = x+2 in y+1 end)
  end

fun countup_from1 (x:int) = 
  let
  fun count (from:int, to:int) = if from=to then to else from::count(from+1,to) in
  count(1,x)
  end

fun countup_from1_better (x:int) = 
  let
  fun count (from:int) = if from=x then x else from::count(from+1)
  in
  count 1
  end

(* max, repeated computation, and options *)
(* looks the same as max1 to clients; implementation avoids valOf *)
fun max2 (xs : int list) = 
  if null xs then NONE else
  let
    fun max_nonempty (xs : int list) = if null (tl xs) then hd xs else
      let
        val tl_ans = max_nonempty(tl xs)
      in
        if hd xs > tl_ans then hd xs else tl_ans
      end
    in
      SOME (max_nonempty xs)
    end

(* benefits of no mutation: recall from lecture 2 *)
(* does not matter if returns an alias *)
fun sort_pair (pr : int×int) = 
  if (#1 pr) < (#2 pr) then pr else (#2 pr, #1 pr)

fun append (xs : int list, ys : int list) = 
  if null xs then ys else (hd xs) :: (append (tl xs, ys))

fun max1 (xs : int list) = 
  if null xs then NONE else
    let
      val tl_ans = max1(tl xs)
    in
      if isSome tl_ans ∧ valOf tl_ans > hd xs then tl_ans else SOME (hd xs)
    end

fun max1 (xs : int list) = 
  if null xs then NONE else
    let
      val tl_ans = max1(tl xs)
      in
        if isSome tl_ans ∧ valOf tl_ans > hd xs then tl_ans else SOME (hd xs)
      end

(* max, repeated computation, and options *)
(* badly named: evaluates to 0 on empty list *)
fun bad_max (xs : int list) = 
  if null xs then 0 else if null (tl xs) then hd xs else
    if hd xs > bad_max(tl xs) then hd xs else bad_max(tl xs)

(* badly named: evaluates to 0 on empty list *)
fun good_max (xs : int list) = 
  if null xs then 0 else if null (tl xs) then hd xs else
    (* for style, could also use a let-binding for (hd xs) *)
    let
      val tl_ans = good_max(tl xs)
      in
        if hd xs > tl_ans then hd xs else tl_ans
      end

fun countup(from : int, to : int) = (* for testing our max functions *) 
  if from=to then to::[] else from::countup(from+1, to)

fun countdown(from : int, to : int) = (* for testing our max functions *) 
  if from=to then to::[]