(** lec 07 in class *)

(** functions can go anywhere other values go *)

fun double x = 2 * x

fun incr x = x + 1

val a_tuple = (double, incr, double(incr 7))

val eighteen = (#1 a_tuple) 9


(* it should *pain* us to write the next three functions separately, but we do not have to *)

fun increment_n_times_lame (n, x) = (* silly example, this is addition (n+x) *)
  if n = 0
    then x
  else 1 + increment_n_times_lame(n-1, x)

fun double_n_times_lame (n, x) =
  if n = 0
    then x
  else 2 * double_n_times_lame(n-1, x)

fun nth_tail_lame (n, xs) =
  if n = 0
    then xs
  else tl (nth_tail_lame(n-1, xs))

(* this is much better: as always, abstract the common pieces into a function n_times(f,n,x) returns f(f(...(f(x))))) where there are n calls to f
note: if we gave x type int, then we could not use this for lists *)

fun n_times (f, n, x) =
  if n = 0
    then x
  else f (n_times(f, n-1, x))

fun increment x = x + 1

val x1 = n_times(increment, 4, 7)

val x2 = n_times(double, 4, 7)

val x3 = n_times(tl, 2, [4, 8, 12, 16])

(* higher-order functions are often polymorphic based on "whatever type of function is passed" but not always: *)

fun times_until_zero (f, x) =
  (* note: a better implementation would be tail-recursive *)
  if x = 0
    then 0
  else 1 + times_until_zero(f, f x)

(* conversely, we have seen polymorphic functions that are not higher-order *)

fun len : 'a list -> int
  | [] => 0
  | x::xs' => 1 + len xs'

(* and we can define functions that use n_times *)
(* motivating and introducing anonymous functions *)

(* fun triple = ... *)

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

(* WILL NOT WORK! *)

n_times (fun triple x = 3 * x, n, x)

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

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

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

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

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

(* Notice the function has no name *)

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

(* here is a very, very useful and common example *)
fun map (f, xs) =  
  case xs of  
    [] ⇒ []  
  | x::xs' ⇒ (f x) :: (map (f, xs'))
val x4 = map ((λ x ⇒ x+1), [4,8,12,16])
val x5 = map (hd, [[1,2],[3,4],[5,6,7]])

(* another very, very useful and common example *)

fun filter (f,xs) =  
  case xs of  
    [] ⇒ []  
  | x::xs' ⇒ if f x then x::(filter (f,xs')) else filter (f,xs')
fun is_even v =  
  (v mod 2 = 0)
fun all_even xs =  
  filter(is_even, xs)

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)

(* Returning a function *)

fun double_or_triple f =  
  if f 7 then λ x ⇒ 2*x  
  else λ x ⇒ 3*x
val dbl = double_or_triple (λ x ⇒ x-3 = 4)
val nine = (double_or_triple (λ x ⇒ x = 42)) 3