fun double x = 2\times 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 \times 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 *)

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(increment,4,7)

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

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

(* 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: *)

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

/* unnecessary function wrapping */

fun nth_tail_poor (n,xs) = n_times((λ y ⇒ tl y), n, xs)

(* bad style: the if e then true else false of functions *)

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

(* good style *)

val rev_poor = λ xs ⇒ List.rev xs

(* good style *)

val rev_good = List.rev

/* here is a very, very useful and common example */
fun map \( (f, xs) \) = 
  \begin{cases} 
  \text{case } xs \text{ of } [] & \Rightarrow [] \\
  x:x': \Rightarrow (f x) : (\text{map } (f, xxs')) 
  \end{cases} 
\]

val x4 = map (\( \lambda x \Rightarrow x+1 \)), \([4,8,12,16] \))

val x5 = map (hd, \([ [1,2], [3,4], [5,6,7] ] \))

\(*\text{ another very, very useful and common example }*\)

fun filter \((f, xs) = \)
  \begin{cases} 
  \text{case } xs \text{ of } [] & \Rightarrow [] \\
  x:x': \Rightarrow \text{if } f x \text{ then } x : (\text{filter } (f, xxs')) \\text{ else } \text{filter } (f, xxs') 
  \end{cases} 
\]

fun is_even v = 
  \(v \mod 2 = 0\)

fun all_even xs = 
  \text{filter(is_even, xs)}

\(*\text{ Higher-order functions over our own datatype bindings }*\)

datatype exp = Constant of int
| Negate of exp
| Add of exp \times exp
| Multiply of exp \times exp

fun true_of_all_constants\((f,e) = \)
  \begin{cases} 
  \text{case of } \text{Constant } i & \Rightarrow f i \\
  \text{Negate } e1 & \Rightarrow \text{true_of_all_constants}(f,e1) \\
  \text{Add}(e1,e2) & \Rightarrow \text{true_of_all_constants}(f,e1) \land \text{true_of_all_constants}(f,e2) \\
  \text{Multiply}(e1,e2) & \Rightarrow \text{true_of_all_constants}(f,e1) \land \text{true_of_all_constants}(f,e2) 
  \end{cases} 
\]

fun all_even_exp e = \text{true_of_all_constants}(\(\lambda x \Rightarrow x \mod 2 = 0\)), e)

\(*\text{ Returning a function }*\)

fun double_or_triple f =
  \begin{cases} 
  \text{if } f ? \text{ then } \lambda x \Rightarrow 2x \\
  \text{else } \lambda x \Rightarrow 3x 
  \end{cases} 
\]

val dbl = double_or_triple (\( \lambda x \Rightarrow x-3 = 4\))

val nine = (double_or_triple (\( \lambda x \Rightarrow x = 42\))) 3