type cart = real × real

datatype shape = Circle of cart × real (* coordinates and radius *)
| Square of cart × real (* coordinates and side length *)
| Rectangle of cart × real × real (* coordinates and side lengths *)

fun area sh =
  case sh of
  Circle(_, r) ⇒ 3.14 × r × r
  Square(_, s) ⇒ s × s
  Rectangle(_, w, l) ⇒ w × l

fun area2 (Circle(_, r)) = 3.14 × r × r
| area2 (Square(_, s)) = s × s
| area2 (Rectangle(_, w, l)) = w × l

(* Here's another nice example showing how pattern matching can be readable, concise, and powerful *)

datatype exp = Constant of int
| Negate of exp
| Add of exp × exp
| Multiply of exp × exp

fun eval (Constant i)  = i
| eval (Add(e1, e2)) = (eval e1) + (eval e2)
| eval (Negate e1)   = ~ (eval e1)
| eval (Multiply(e1, e2)) = (eval e1) × (eval e2)

(* Tail Recursion *)

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

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

fun inc_all1 xs =
  case xs of
  [] ⇒ []
  | x :: xs ⇒ (x + 1) :: inc_all1 xs

fun inc_all2 xs =
  let fun aux (acc, xs) =
    case xs of
    [] ⇒ rev2 acc (* need to reverse! still constant space!!! *)
    | x :: xs ⇒ aux (x + 1 :: acc, xs)
  in aux ([], xs)
  end

fun repeat1 (x, n) =
  if n = 0 then []
  else x :: (repeat1(x, n - 1))

fun repeat2 (x, n) =
  let fun aux (acc, i) =
    if i = 0 then acc
    else aux(x :: acc, (i - 1))
  in aux([], n)
  end

(* range(lo, hi) returns a list of all values from lo to (hi - 1) *)

fun range (lo, hi) =
  let fun aux (acc, i) =
    if i < lo then acc
    else aux(i :: acc, (i - 1))
  in aux([], (hi - 1))
  end

(* (pair_chain l) returns a list of all pairs of consecutive elements in l *)

fun pair_chain l =
  let fun aux (acc, i) =
    if i < length l then aux((l[i], l[i+1]) :: acc, (i + 1))
    else rev2 acc
  in aux([], 0)
  end

(* triples(xs, ys, zs) combines three lists into triple list if they have equal length otherwise raise a LengthMismatch exception *)

exception LengthMismatch

fun triples (xs, ys, zs) =
  let fun aux (m :: ms, n :: ns, p :: ps, acc) = aux(ms, ns, ps, ((m, n, p) :: acc))
      | aux (_, _, _) = raise LengthMismatch
  in aux(xs, ys, zs, [])
  end

(* (choose2 l) returns a list of pairs using all combination of elements of l *)

fun choose2 [] = []
| choose2 (x :: xs) =
  let fun choose_first (first, []) = []
      | choose_first (first, x :: xs') = (first, x) :: choose_first(xs', choose2 xs)
  in append(choose_first(x, xs), (choose2 xs))
  end

fun choose2_tail l =
  let fun choose_first (first, []) = acc
      | choose_first (first, x :: xs') = choose_first(xs', acc')
  in choose_first(l, [])
  end

fun repeat (x, n) =
  if n = 0 then []
  else x :: (repeat(x, n - 1))
choose_second(xs', acc)
  in
  choose_first(l, [])
end