Good and bad of Scheme

An advanced cognitive task:
1. Remember
2. Understand
3. Apply
4. Analyze
5. Evaluate
6. Create
fun length []     = 0
               | length (x::xs) = 1 + length xs

val res = length [1,2,3]
fun map f [] = []
   | map f (x::xs) = (f x) :: (map f xs)

val res1 = map length [[], [1], [1,2], [1,2,3]]
fun map f [] = []
| map f (x::xs) = f x :: map f xs

val res1 = 
map length [[] , [1] , [1,2], [1,2,3]]
Filter

fun filter pred [] = []
  | filter pred (x::xs) = (* no 'pred?' *)
    let val rest = filter pred xs
    in  if pred x then
      (x :: rest)
    else
      rest
    end

val res2 =
  filter (fn x => (x mod 2) = 0) [1,2,3,4]
fun filter pred [] = []
  | filter pred (x::xs) = (* no 'pred?' *)
      let val rest = filter pred xs
      in  if pred x then
        x :: rest
        else
        rest
    end

val res2 = filter (fn x => (x mod 2) = 0) [1,2,3,4]
fun exists pred [] = false
  | exists pred (x::xs) =
    (pred x) orelse (exists pred xs)

val res3 =
  exists (fn x => (x mod 2) = 1) [1,2,3,4]
(* Note: fn x => e is syntax for lambda *)
fun exists pred [] = false
  | exists pred (x::xs) =
      pred x orelse exists pred xs

val res3 =
  exists (fn x => (x mod 2) = 1) [1,2,3,4]
(* Note: fn x => e is syntax for lambda *)
fun all pred []       = true
    | all pred (x::xs) =
        (pred x) andalso (all pred xs)

val res4 = all (fn x => (x >= 0)) [1,2,3,4]
fun all pred [] = true
  | all pred (x::xs) =
      pred x andalso all pred xs

val res4 = all (fn x => (x >= 0)) [1,2,3,4]
exception TooShort

fun take 0 _ = [] (* wildcard! *)
| take n [] = raise TooShort
| take n (x::xs) = x :: (take (n-1) xs)

val res5 = take 2 [1,2,3,4]
val res6 = take 3 [1]
    handle TooShort =>
        (print "List too short!"; [])

(* Note use of exceptions. *)
Take, without redundant parentheses

exception TooShort

fun take 0 _       = [] (* wildcard! *)
    | take n []       = raise TooShort
    | take n (x::xs)  = x :: take (n-1) xs

val res5 = take 2 [1,2,3,4]
val res6 = take 3 [1]
    handle TooShort =>
        (print "List too short!"; [])

(* Note use of exceptions. *)
fun drop 0 zs = zs
| drop n [] = raise TooShort
| drop n (x::xs) = drop (n-1) xs

val res7 = drop 2 [1,2,3,4]
val res8 = drop 3 [1]
    handle TooShort =>
    (print "List too short!"; [])
fun takewhile p [] = []
    | takewhile p (x::xs) =
        if p x then (x :: (takewhile p xs))
        else []
    
fun even x = (x mod 2 = 0)
val res8 = takewhile even [2,4,5,7]
val res9 = takewhile even [3,4,6,8]
Takewhile, without redundant parentheses

fun takewhile p [] = []
  | takewhile p (x::xs) =
    if p x then x :: takewhile p xs
    else []

fun even x = (x mod 2 = 0)
val res8 = takewhile even [2,4,5,7]
val res9 = takewhile even [3,4,6,8]
fun dropwhile p [] = []
  | dropwhile p (zs as (x::xs)) = if p x then (dropwhile p xs) else zs
val res10 = dropwhile even [2,4,5,7]
val res11 = dropwhile even [3,4,6,8]

(* fancy pattern form: zs as (x::xs) *)
Dropwhile, without redundant parentheses

fun dropwhile p [] = []
  | dropwhile p (zs as (x::xs)) = 
      if p x then dropwhile p xs else zs

val res10 = dropwhile even [2,4,5,7]
val res11 = dropwhile even [3,4,6,8]

(* fancy pattern form: zs as (x::xs) *)
Folds

fun foldr p zero [] = zero
  | foldr p zero (x::xs) = p (x, (foldr p zero xs))

fun foldl p zero [] = zero
  | foldl p zero (x::xs) = foldl p (p (x, zero)) xs

val res12 = foldr (op +) 0 [1,2,3,4]
val res13 = foldl (op * ) 1 [1,2,3,4]

(* Note 'op' to use infix operator as a value *)
Folds, without redundant parentheses

fun foldr p zero [] = zero |
  foldr p zero (x::xs) = p (x, foldr p zero xs )

fun foldl p zero [] = zero |
  foldl p zero (x::xs) = foldl p (p (x, zero)) xs

val res12 = foldr (op+) 0 [1,2,3,4]
val res13 = foldl (op*) 1 [1,2,3,4]

(* Note 'op' to use infix operator as a value *)
ML—Five Questions

Values: num/string/bool, constructed data

Syntax: definitions, expressions, patterns, types

Environments: names stand for values (and types)

Evaluation: uScheme + case and pattern matching

Initial Basis: medium size; emphasizes lists

(Question Six: type system—a coming attraction)
A note about books

Ullman is easy to digest

Ullman costs money but saves time

Ullman is clueless about good style

Suggestion:
• Learn the syntax from Ullman
• Learn style from Ramsey, Harper, & Tofte

Details in course guide _Learning Standard ML_