CSE 341 Section Handout #3
Cheat Sheet

Higher-Order Functions (5.4)

(* post: Returns a list where f is applied to each element of the list *)
fun map(f, []) = []
| map(f, x::xs) = f(x) :: map(f, xs);

(* post: Returns a list of elements from the given list that satisfy the given predicate f *)
fun filter(f, []) = []
| filter(f, x::xs) =
  if f(x) then x :: filter(f, xs)
  else filter(f, xs);

(* post: Returns a value that is the two valued function f applied to every two values in the list *)
fun reduce(f, [x]) = x
| reduce(f, x::xs) = f(x, reduce(f, xs));

Exceptions (5.2)

(* Declaring an exception type *)
exception name;

(* Throwing (raising) an exception *)
raise name

(* Handling (catching) an exception; tries to compute expression1, but if it throws the given kind of exception, instead produces expression2 *)
expression1 handle exception => expression2

Example:
exception Negative;
fun factorial(0) = 1
| factorial(n) = if n < 0 then raise Negative
  else n * factorial(n - 1);

Composition of Functions (5.6)

function1 o function2

The o operator does composition (combination) of functions exactly like you would write it in Mathematics, i.e. h(x) = f(g(x)) = (f o g) (x).

Example:
Computes the square roots of all integers between 1 and 100 inclusive. Using the higher-order function map showing the transformation from one form to another.

map(round, map(Math.sqrt, map(real, 1--100)));
map(round o Math.sqrt o real, 1--100);

Anonymous Functions (5.1.3)

fn parameter(s) => expression

Defining Infix Operators (9.1.4)

infix operator;
fun param operator param = expression;

Example:
infix --;
fun min -- max =
  if min > max then []
  else min :: (min+1 -- max);
Higher-Order Functions; Anonymous Functions

1. Define a function `isPrime` that takes an integer parameter and returns `true` iff the integer is prime. The approach you should use is to verify that it has no factors in the range of 2 through its square root. First write it with a helper function `isFactor`, then write a second version using an anonymous function.

2. Define a function `sumOfSquares` that takes a list of integers as a parameter and returns the sum of the squares of the integers in the list. For example, `sumOfSquares([3, 4, 9])` should return $3^2 + 4^2 + 9^2 = 106$. Write it as a one-line function using `map/filter/reduce` and anonymous functions.

3. Define a function `sumOfSquares2` that takes an integer $n$ as a parameter and returns the sum of the squares of the integers 1 through $n$ inclusive. For example, `sumOfSquares2(5)` should return $1^2 + 2^2 + 3^2 + 4^2 + 4^2 = 55$. Write it as a one-line function using `map/filter/reduce` and anonymous functions.

4. Define a function called `oddProduct` that takes an integer $n$ as a parameter and returns the product of the first $n$ odd numbers. Write it as a one-line function using `map/filter/reduce` and anonymous functions.

5. Define a function `len` that computes the length of a list. Write it as a one-line function using `map/filter/reduce` and anonymous functions.

6. Using `map/filter/reduce`:
   (a) Use `map` to do the following:
      - Change every lowercase letter in a list of characters to the corresponding uppercase letter. Do not assume that only the lowercase letters appear in the list.
      - Truncate each string in a list of strings so that it is no more than 5 characters long, that is, delete the 6th and subsequent characters while leaving shorter strings alone.
   (b) Use `filter` to do the following:
      - Find those elements of a list of strings that begin with the character "a".
      - Find those elements of a list of strings that are at most 3 characters long.
   (c) Use `reduce` to do the following:
      - Find the logical or of a list of booleans
      - Find the maximum of a list of `reals`.

Composition of Functions

7. Define a function named `squareWhole` that accepts a list of real numbers and produces the squares of the whole-number portions of those numbers. That is, you must throw away any portion of each real number after the decimal point, then square it. Write your function as a one-line definition using composition of functions and higher-order functions. For example, if `numbers` stores `[3.4, 1.7, 5.8, 10.6]`, `squareWhole(numbers)` should produce `[9.0, 1.0, 25.0, 100.0]`. Note that the elements of the list produced are real numbers and not integers.
Curried Functions; Function Composition

8. Define the following functions using val declarations with curried functions and the function composition operator. Do not define any helper functions using fun declarations or the fn anonymous function notation.

   (a) Function double that takes an int and returns its double (the integer twice as large in value)
   (b) Function prependStar that takes a string as a parameter and that returns a new string with a star ("*") followed by the parameter. For example, prependStar("hello") should return "*hello".
   (c) Function oneTo that takes an integer \( n \) and returns the list of integers from 1 to \( n \) inclusive.
   (d) Function addOne that takes a list of integer values and that returns the list obtained by adding one to each number in the list.
   (e) Function \( f \) that takes an integer \( n \) and that returns the absolute value of \( 2n + 10 \).
   (f) Function primeProduct that takes a list of integers as a parameter and that returns the product of the primes in the list (you can use function isPrime from problem #1).

9. Each of the following curried definitions is flawed because it needs parentheses. Indicate how ML will group the items and where parentheses need to be added:

   (a) fun f c:char = 1.0
   (b) fun f x::xs = []
   (c) print Int.toString 123
   (d) val add2 = map2 curry op+ 2
1. 
   fun isPrime(n) =
       let fun isFactor(m) = n mod m = 0
           val factors = filter(isFactor, 2--trunc(Math.sqrt(real(n))))
       in n > 1 andalso factors = []
       end;
   
   fun isPrime(n) =
       n > 1 andalso
       filter(fn(x) => n mod x = 0, 2--trunc(Math.sqrt(real(n)))) = [];

2. 
   (* sum of squares of values from a list *)
   fun sumOfSquares(lst) = reduce(op +, map(fn x => x * x, lst));

3. 
   (* sum of squares of 1 through n *)
   fun sumOfSquares2(n) = reduce(op +, map(fn x => x * x, 1--n));

4. 
   (* product of first n odd numbers *)
   fun oddProduct(n) = reduce(op *, map(fn x => 2 * x + 1, 1--n));

5. 
   (* length of a list *)
   fun len(lst) = reduce(op +, 0::map(fn x => 1, lst));

6. (no solution provided)

7. 
   fun squareWhole(lst) = map(real o (fn(x) => x*x) o trunc, lst);
8.
(a) val double = curry op* 2;
(b) val prependStar = curry op ^ "*";
(c) val oneTo = curry op-- 1;
(d) val addOne = map2 (curry op+ 1);
(e) val f = abs o curry op+ 10 o curry op* 2;
(f) val primeProduct = reduce2 op* o filter2 isPrime;

9.
(a) original code: fun f c:char = 1.0
    is interpreted as: fun (f c):char = 1.0
    it should be: fun f(c:char) = 1.0

(b) original code: fun f x::xs = []
    is interpreted as: fun (f x)::xs = []
    it should be: fun f(x::xs) = []

(c) original code: print Int.toString 123
    is interpreted as: (print Int.toString) 123
    it should be: print(Int.toString 123)

(d) original code: val add2 = map2 curry op+ 2
    is interpreted as: val add2 = (map2 curry) op+ 2
    it should be: val add2 = map2 (curry op+ 2)