- The `fn` expression for creating anonymous functions (using matches).
- The idea of higher-order functions.
- The idea of currying.
- The long and short forms for writing curried functions.
- The predefined, curried, higher-order functions `map`, `foldr`, and `foldl`.

Exercises

The first 25 exercises should all have one-line solutions using `map`, `foldr`, or `foldl`. You can also use other predefined functions, of course, but *do not write any additional named functions and do not use explicit recursion.* If you need helper functions, use anonymous ones. For example, if the problem says "write a function `add2` that takes an `int list` and returns the same list with 2 added to every element," your answer should be

```
fun add2 x = map (fn a => a + 2) x;
```

You have seen some of these problems before. The trick now is to solve them in this new, concise form.

Exercise 1 Write a function `il2rl` of type `int list -> real list` that takes a list of integers and returns a list of the same numbers converted to type `real`. For example, if you evaluate `il2rl [1,2,3]` you should get `[1.0,2.0,3.0]`.

Exercise 2 Write a function `ordlist` of type `char list -> int list` that takes a list of characters and returns the list of the integer codes of those characters. For example, if you evaluate `ordlist [#"A",#"b",#"C"]` you should get `[65,98,67]`.

Exercise 3 Write a function `squarelist` of type `int list -> int list` that takes a list of integers and returns the list of the squares of those integers. For example, if you evaluate `squarelist [1,2,3,4]` you should get `[1,4,9,16]`.

Exercise 4 Write a function `multpairs` of type `(int * int) list -> int list` that takes a list of pairs of integers and returns a list of the products of each pair. For example, if the input is `[(1,2),(3,4)]`, your function should return `[2,12]`.

Exercise 5 Write a function `inclist` of type `int list -> int -> int list` that takes a list of integers and an integer increment, and returns the same list of integers but with the integer increment added to each one. For example, if you evaluate `inclist [1,2,3,4] 10` you should get `[11,12,13,14]`. Note that the function is curried.

Exercise 6 Write a function `sqsum` of type `int list -> int` that takes a list of integers and returns the sum of the squares of those integers. For example, if you evaluate `sqsum [1,2,3,4]` you should get `30`.

Exercise 7 Write a function `bor` of type `bool list -> bool` that takes a list of boolean values and returns the logical OR of all of them. If the list is empty, your function should return `false`.

Exercise 8 Write a function `band` of type `bool list -> bool` that takes a list of boolean values and returns the logical AND of all of them. If the list is empty, your function should return `true`.

Exercise 9 Write a function `bxor` of type `bool list -> bool` that takes a list of boolean values and returns the logical exclusive OR of all of them. (It should return `true` if the number of `true` values in the list is odd and `false` if the number of `true` values is even.) If the list is empty, your function should return `false`.

Exercise 10 Write a function `dupList` of type `'a list -> 'a list` whose output list is the same as the input list, but with each element of the input list repeated twice in a row. For example, if the input list is `[1,3,2]`, the output list should be `[1,1,3,3,2,2]`. If the input list is `[]`, the output list should be `[]`.

Exercise 11 Write a function `mylength` of type `'a list -> int` that returns the length of a list. (Of course, you may not use the predefined `length` function to do it.)

Exercise 12 Write a function `il2absrl` of type `int list -> real list` that takes a list of integers and returns a list containing the absolute values of those integers, converted to real numbers.

Exercise 13 Write a function `truecount` of type `bool list -> int` that takes a list of boolean values and returns the number of trues in the list.