*Exercise 2* Write a function `cuber` of type `real -> real` that returns the cube of its parameter.

*Exercise 3* Write a function `fourth` of type `'a list -> 'a` that returns the fourth element of a list. Your function need not behave well on lists with less than four elements.

*Exercise 4* Write a function `min3` of type `int * int * int -> int` that returns the smallest of three integers.

*Exercise 5* Write a function `red3` of type `'a * 'b * 'c -> 'a * 'c` that converts a tuple with three elements into one with two by eliminating the second element.

*Exercise 6* Write a function `thirds` of type `string -> char` that returns the third character of a string. Your function need not behave well on strings with lengths less than 3.

*Exercise 7* Write a function `cycle1` of type `'a list -> 'a list` whose output list is the same as the input list, but with the first element of the list moved to the end. For example, `cycle1 [1,2,3,4]` should return `[2,3,4,1]`.

*Exercise 8* Write a function `sort3` of type `real * real * real -> real list` that returns a list of three real numbers, in sorted order with the smallest first.

*Exercise 9* Write a function `del3` of type `'a list -> 'a list` whose output list is the same as the input list, but with the third element deleted. Your function need not behave well on lists with lengths less than 3.

*Exercise 10* Write a function `sqsum` of type `int -> int` that takes a non-negative integer *n* and returns the sum of the squares of all the integers 0 through *n*. Your function need not behave well on inputs less than zero.

*Exercise 11* Write a function `cycle` of type `'a list * int -> 'a list` that takes a list and an integer n as input and returns the same list, but with the first element cycled to the end of the list n times. (Make use of your `cycle1` function from a previous exercise.) For example, `cycle ([1,2,3,4,5,6],2)` should return the list `[3,4,5,6,1,2]`.



*Exercise 12* Write a function `pow` of type `real * int -> real` that raises a real number to an integer power. Your function need not behave well if the integer power is negative.

*Exercise 13* Write a function `max` of type `int list -> int` that returns the largest element of a list of integers. Your function need not behave well if the list is empty. *Hint:* Write a helper function `maxhelper` that takes as a second parameter the largest element seen so far. Then you can complete the exercise by defining

```
fun max x = maxhelper (tl x, hd x);
```

*Exercise 14* Write a function `isPrime` of type `int -> bool` that returns true if and only if its integer parameter is a prime number. Your function need not behave well if the parameter is negative.

*Exercise 15* Write a function `select` of this type:

```
'a list * ('a -> bool) -> 'a list
```

that takes a list and a function *f* as parameters. Your function should apply *f* to each element of the list and should return a new list containing only those elements of the original list for which *f* returned true. (The elements of the new list may be given in any order.) For example, evaluating `select ([1,2,3,4,5,6,7,8,9,10], isPrime)` should result in a list like `[7,5,3,2]`. This is an example of a *higher-order* function, since it takes another function as a parameter. We will see much more about higher-order functions in Chapter 9.

## Further Reading

This is a great book to help you learn more about ML:

> Ullman, Jeffrey D. *Elements of ML Programming*. Upper Saddle River, NJ: Prentice Hall, 1998.

It covers the basics seen in this chapter, the more advanced things that will be seen in later chapters, and the even more advanced things there will not be time to discuss.