Chapter 4

Functions

It is theoretically possible to write any program using only loops, conditional and assignment statements. In practice, it is difficult to write any program much longer than a page using only these statements. We need a tool that will allow us to assemble a large program from smaller pieces. That is the role of functions in programming. Once you have mastered functions and the art of programming with them you can write programs to implement almost any algorithm you can think of.

4.1 Concepts

A *function* is a block of code that has a name attached to it. We execute this code by *calling* the function. In Python we call a function by giving its name, followed by open and closed parentheses. For example, if a function is named foo, we call it with foo(). We may put values inside the parentheses; this is the way inputs are given to the function. We call values put in the parentheses *arguments* to the function. In the expression

```
bar(3, 5)
```

a function named bar is being called with arguments 3 and 5.

There are two kinds of functions: functions that return values and functions that do not return values. These two kinds of functions are used in very different ways. When a function *returns* a value, you can use a call to this function as a placeholder for the value the function returns. For example, we might have a function addOne that takes a numeric argument and returns that argument plus 1. This means that addOne(5) returns the value 6. We might have a line of a program x = addOne(5). The effect of this line is to set variable x to 6. The call to function addOne produces a value and this call can be used anywhere we can use that value. We could even use it as an argument for another call to addOne: y = addOne(addOne(5)) sets variable y to 7. When a function does not return a value we use a call to it as a statement by itself. For example, we might have a function printCalendar that takes a month and a year as arguments and prints a calendar for that month. We could call this function as printCalendar(9, 2011) to print a calendar for September, 2011. We would not say x = printCalendar(9, 2011) because this function does not return a value.

When writing about functions it is customary to put parentheses after the name of the function to emphasize the fact that this is a function. Unless there is a reason to discuss the arguments, we often omit them from the name. In the examples above, we might discuss the functions addOne() and printCalendar(). We have already used a number of standard Python functions. **input**() is a function that takes an optional string argument and returns a string typed by the user. **range**() is a function that takes one, two, or three integer arguments and returns a list of numbers. **len**() is a function that takes a sequence and returns its length. We have seen a function **print**() that prints its argument (and doesnt return anything). Note that in all of these examples we can use the function without knowing how it is written. All that matters is what inputs the function takes and what value, if any, it returns. One great advantage of functions is that once they are written we can use them without thinking about their details, only about their inputs and outputs. This is what allows us to write long programs. Functions *encapsulate* a process so that we may concentrate on what it does rather than how it works.