2
A Cook's tour of OCaml

To give a flavor of working with the OCaml programming language, we introduce OCaml through an INTERPRETER of the language, called `ocaml`, which is invoked from the command line thus:[1]

```
% ocaml
```

Upon running `ocaml`, you will see a PROMPT ("#") allowing you to type an OCaml expression.

```
% ocaml
        OCaml version 4.11.1
#
```

Exercise 1

The startup of the `ocaml` interpreter indicates that this is version 4.11.1 of the software. What version of `ocaml` are *you* running?

Once the OCaml prompt is available, you can enter a series of OCaml expressions to calculate the values that they specify. Numeric (integer) expressions are a particularly simple case, so we'll start with those. The integer LITERALS – like `3` or `42` or `-100` – specify integer values directly, but more complex expressions built by applying arithmetic functions to other values do as well. Consequently, the OCaml interpreter can be used as a kind of calculator.

```
# 42 ;;
- : int = 42
# 3 + 4 * 5 ;;
- : int = 23
# (3 + 4) * 5 ;;
- : int = 35
```

Since this is the first example we've seen of interaction with the OCaml interpreter, some glossing may be useful. The OCaml interactive prompt, '#', indicates that the user can enter an OCaml expression, such as '`3 + 4 * 5`'. A double semicolon '`;;`' demarcates the end of the expression. The system *reads* the expression, *evaluates* it (that

[1] We assume that you've already installed the OCaml tools, as described at the `ocaml.org` web site.

is, calculates its value), and *prints* an indication of the result, then *loops* back to provide another prompt for the next expression. For this reason, the OCaml interactive system is referred to as the "READ-EVAL-PRINT LOOP" or REPL.[2] Whenever we show the results of an interaction with the REPL, the interpreter's output will be shown in a *slanted font* to distinguish it from the input.

[2] To exit the REPL, just enter the end-of-file character, `^d`, typed by holding down the control key while pressing the `d` key.

You'll notice that the REPL obeys the standard order of operations, with multiplication before addition for instance. This precedence can be overwritten in the normal manner using parentheses.

Exercise 2

Try entering some integer expressions into the OCaml interpreter and verify that appropriate values are returned.

Although we'll introduce the aspects of the OCaml language incrementally over the next few chapters, to get a general idea of using the language, we demonstrate its use with the GCD algorithm from Chapter 1. We type the definition of the `gcd_euclid` function into the REPL:

```
# let rec gcd_euclid a b =
#   if b = 0 then a
#   else gcd_euclid b (a mod b) ;;
val gcd_euclid : int -> int -> int = <fun>
```

Now we can make use of that definition to calculate the greatest common divisor of 20 and 28

```
# gcd_euclid 20 28 ;;
- : int = 4
```

But we're getting ahead of ourselves.