Side-effect free functions

Side effects are operations that change the global state of a computation. Formally, all assignments and all input/output operations are considered side-effects. The term *functional programming* (or more accurately *purely functional programming*) refers to a style of programming where functions have no side effects.

While it is impossible to write any real applications without any side effects (you could never write anything to the user, and you could never save anything to disk), functional programming is still a very important way of programming large parts of an application.

We can get most of the benefits of functional programming while still allowing some kinds of side effects, for instance:

- input/output that does not effect the global state of the system, for instance output to the screen,
- assignments to local variables that do not survive the execution of the function in which they are located.

In other words, we want the execution of a function to have *no effect on the global state of the system*. If, in addition, the function is autonomous, we can be sure that repeated executions of the same function with the same arguments gives exactly the same result.

If the function is not without side effects, the following can happen:

```
> (setq *l* (list 1 2 3 4 5))
(1 2 3 4 5)
> (defun fun (l)
    (setq *l* (cdr l))
    (length l))
fun
> (fun *l*)
5
> (fun *l*)
4
> (fun *l*)
3
```

Notice that `fun` is by definition autonomous, but we lose because it has a side effect on the global variable `*l*`. If instead, we had written: > (defun fun (l) (setq *l* (cdr l)) (length l)) fun > (fun '(1 2 3 4 5)) 5 > (fun '(1 2 3 4 5)) 5 > (fun '(1 2 3 4 5)) 5