Function calls

The order in which arguments of function calls are evaluated is undefined (as in C++ but contrary to Java). For instance, in

```
f(g(),h())
```

the order of evaluation of `g()` and `h()` is undefined, i.e.,, the programmer has no guarantee whether `g()` will be evaluated before or after `h()`.

The order of evaluation of elements in a list is also undefined. For instance

```
[g(), h()]
```

Functions/Modules

Modules in Erlang are a database of functions.

The module name must be the basename of the file containing it, and the extension of the file must be `erl`. So, file `foo.erl` contains the Erlang module `foo`, which is specified by the first line

```
-module(foo).
```

The purpose of Erlang modules is to restrict the visibility of the function definitions, since many functions have a merely technical, internal, purpose and are not directly related to the higher-level of module functionalities or services. Exported functions need to be listed, together with their arity. For example

```
-module(foo).
-export([fact/1]).
```

Functions/Qualified calls

Functions can be called in two ways, either by qualifying their name or not. Qualification means writing the name of the module the called function belongs to. For instance

```
math:fact(4)
```

is a qualified call, whereas

```
fact(N-1)
```

is a non-qualified call.

When the caller and the called function are in different modules, the qualified form is mandatory. When calling a function which is in the same module as the caller, the two forms are allowed. There is actually a difference we shall discuss later, but which is, at this point, irrelevant.

Functions/Recursivity

Recursivity consists in a function calling itself, just like the factorial function did at page 18. The function call in the body of the second case is a recursive call.

This technique is the only way in Erlang to implement the equivalent of loops in other imperative programming languages, i.e., in non-functional programming languages.

Indeed, since there are no mutable variables in Erlang, i.e., variables whose value can be changed, a loop would be pointless.

The combination of matching and recursivity is the heart of Erlang programming.