Terms/Atoms

Let us define more precisely what kind of data Erlang operates on. These data are called **terms**.

Some terms are only identified by their name, called an **atom**. An atom starts with a lower-case letter which can be followed by a series of characters out of lower-case letters, upper-case letters, digits and the underscore character ('_').

Some atoms can be **quoted** by an opening single-quote and a closing one. In this case, it may contain blank characters. For example

```
anna    apha_beta_proc  x_25    'This is a quoted atom'
x25     call_Java       x_25AB   x_  x____y
```

Function names are atoms.

Terms/Quoted atoms

Quoted atoms look like some strings in some languages, like Bash, but they are very different they cannot be modified. Therefore, they are like constant character strings.

In some programming languages, a character string can be modified in place, but quoted atoms do not allow this.

Terms/Numbers

Numbers. in Erlang include integer numbers and floating-point numbers. The syntax of integer is as expected, for example

```
1      1234      0      -97
```

The lower and larger integers are limited by the actual Prolog system in use.

Floating-point numbers follow the usual syntax too, like

```
100     -7     3.14     -0.06     100.5     1.5e-3
```

Atoms and numbers define the group of **constants**.

Terms/Numbers (cont)

In Erlang, there are no characters. Instead, they can be handled throughout their ASCII code.

There is a special operator `$` which returns the ASCII code for a given character. For example `$A` evaluates to 65.

It is possible to input integers in a base which is not 10 by using a special notation #. For example `16#ffff` represents 65535 (in base 10).

This operator works only if the base ranges from 2 to 16.

Terms/Variables

A **variable** is a name for a term, but, contrary to atoms, variables do not define any object. For example

```
X = 25
```

does not define the constant term 25 but do give it the name X. The term 25 is defined just by being written, just like atoms.

One variable denotes a term in a set. For example, in

```
fact(0) -> 1;
fact(N) -> N * fact(N-1).
```

the variable N denotes any number which is not the zero integer.

Terms/Variables (cont)

They must start with an upper-case letter and may be followed by any number of letters, digit and underscores, in any order. For example, the following are valid variables:

```
X  Obj_List  Object2  Result  ObjList  X_  Obj__
```

If a variable appears only once in the head of a clause and not in the body, it is an **unknown variable** and can be replaced by an underscore. First, consider

```
-module(bool).
-export([f/2]).
f(true,true)   -> false;
f(true,false)  -> true;
f(false,true)  -> true;
f(false,false) -> false.
```

Terms/Variables (cont)

It is equivalent to

```
-module(bool2).
-export([f/2]).
f(X,X) -> false;
f(X,Y) -> true.
```

and

```
-module(bool1).
-export([f/2]).
f(X,X) -> false;
f(_,_) -> true.
```

Important: When several underscores occur in the same head, each one denotes a different term, in general.

Terms/ Variables/Lexical scoping

Given an occurrence of a variable, the part of the program where this variable is usable, or bound, is called the **scope**.

Erlang uses **lexical scoping**, which means

- the same variable always represents the same term inside a clause;
- the same variable in two different clauses represent different terms, in general.

More about the scope at page 52.

Terms/Tuples

Some terms can be linearly compounded into one term, called a **tuple**.

The number of components of a tuple is called arity. The syntax of tuples is different from mathematics in that it requires curly braces instead of parentheses:

```
{a, 12, 'hello'}
{1, 2, {3, 4}, {a, {b, {c}}}}
{}
```

Note that tuples can

- contain terms of different kinds,
- be embedded,
- be empty.

Terms/Lists

A **list** is a term made of a series of other terms. The terms are separated by commas and enclosed between an opening square bracket and a closing one. For example

```
[1, abc, [12], 'foo bar']
[]
[a, b, c]
```

are lists. Note that terms of different kinds can be mixed. Lists of integers which denote ASCII characters can be represented with a shorthand, e.g. `"abc"` is equivalent to `[$a,$b,$c]`, that is: `[97,98,99]`.

Note that in C++ also, there is no string type.

Terms/Lists (cont)

It is often useful to refer to the first element of a list. That is why Erlang provides a special notation for lists that allows to distinguish the first elements, called the **head**, and to collapse the remaining elements into a list called the **tail**, using the notation `[` *Head* `|` *Tail* `]`.

For example

```
[1, abc, {4,5}, "hello"]
[1, abc, {4,5}, [$h,$e,$l,$l,$o]]
[1, abc, {4,5} | ["hello"]]
[1, abc | [{4,5}, "hello"]]
[1 | [abc, {4,5}, "hello"]]
```

are different ways to denote the same list.

Terms/Lists (cont)

Note that

```
[1, abc, {4,5} | "hello"]
```

is **not** equivalent to the previous lists. It is in fact equivalent to

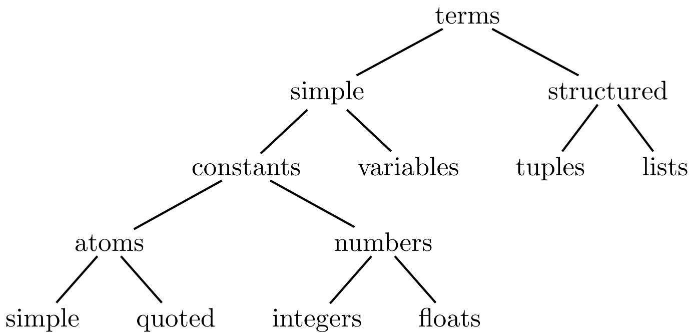
```
[1, abc, {4,5}, $h, $e, $l, $l, $o]
```

Note also that

```
[1, abc | {4,5}, "hello world"]
```

is **invalid** because the tail must be *one* single term.

Terms/Lists (cont)



Ground terms, values and expressions

It is sometimes useful to distinguish between two kinds of terms: **ground terms** and **non-ground terms**.

Ground terms are terms that do not contain any variable.

An **expression** is a syntactic construct involving function calls and terms. For example `5 * fact(5-1)` is an expression but not a term, whereas `X` and `7` are both and expression and a term. The evaluation of an expression does not terminate or lead to a **value** or an error.

A value is thus a ground term that cannot be further computed. For example, 25 is a value but `2 + 5` is not (it is an expression).