Erlang v5.7 - Syntax Card (05 Aug 2009)

mailto:ariel.ortiz@itesm.mx *http://www.arielortiz.com/*

Comments
Start with a percent symbol (`%`) and extend to the end of line. There are no block comments.

Variables
Start with an uppercase letter.

Functions
```
name(pattern,…) [when guard] -> expr,…;
   …
name(pattern,…) [when guard] -> expr,….
```

Anonymous Functions
```
fun
   (pattern,…) [when guard] -> expr,…;
   …
   (pattern,…) [when guard] -> expr,…
end
fun fun_name/arity
fun module_name:fun_name/arity
```

Case
```
case expr of
   pattern [when guard] -> expr,…;
   …
   pattern [when guard] -> expr,…
end
```

If
```
if
   guard -> expr,…;
   …
   guard -> expr,…
end
```

Receive
```
receive
   pattern [when guard] -> expr,…;
   …
   pattern [when guard] -> expr,…
[after milliseconds -> expr,…]
end
```

Try/ Catch
```
try expr,… [of
   pattern [when guard] -> expr,…;
   …
   pattern [when guard] -> expr,…]
catch
   [type:] pattern [when guard] -> expr,…;
   …
   [type:] pattern [when guard] -> expr,…
[after expr,…]
end
```
type must be: `throw`, `exit`, `error`

These are generated by one of these functions:
```
   throw(Why)
   exit(Why)
   erlang:error(Why)
```

Module Attributes
```
-module(Module).
-export([FunName/Arity,…]).
-import(Module,
  [FunName/Arity,…]).
-compile(export_all).
```

Defining Macros
```
-define(Const,Replacement).
-define(Func(Var,…),
  Replacement).
```

Predefined Macros
```
?MODULE
?FILE
?LINE
```

Marco Directives
```
-undef(Macro).
-ifdef(Macro).
-ifndef(Marco).
-else.
-endif.
```

Data Types	*Notes*	*Examples*
integer	Bignums supported.	`42 $A 2#1010`
float	IEEE 754 64-bit.	`3.14 0.99e-10`

atom	Must start with lower-case letter, or be enclosed in single quotes.	`apple`
		`'Hello World'`
		`true false`

tuple	Fixed number of items.	`{1,mango,9.9}`

list	Variable number of items. Implemented as linked lists.	`[a, b, c]`
		`[a|[b|[c|[]]]]`
		`"Hello"`
		`[$H,$e,$l,$l,$o]`

binary	Untyped contiguous memory region.	`<<7,6,10>>`
		`<<"Hello">>`

List Comprehensions

[*expr* **||** *qualifier*, *qualifier*, … **]**

Where *qualifier* can be:

- A generator: *pattern* **<-** *list_expr*
- A filter: any boolean expression

Example:
```
[X*X||X<-[3,-1,2,4],X>0]
⇒ [9,4,16]
```

Complete Erlang Documentation
http://www.erlang.org/doc/

	Operator	*Description*	*Examples*
1	+	Unary plus	`+12 ⇒ 12`
	-	Unary minus	`-12 ⇒ -12`
	bnot	Unary bitwise not	`bnot 1 ⇒ -2`
	not	Unary logical not	`not true ⇒ false`
2	*****	Multiplication	`4*5 ⇒ 20`
	/	Floating point division	`20/6 ⇒ 3.33333`
	div	Integer division	`20 div 6 ⇒ 3`
	rem	Integer remainder	`20 rem 6 ⇒ 2`
	and	Logical and	`true and true ⇒ true`
	band	Bitwise and	`5 band 3 ⇒ 1`
3	+	Plus	`4+5 ⇒ 9`
	-	Minus	`4-5 ⇒ -1`
	bor	Bitwise or	`5 bor 3 ⇒ 7`
	bxor	Bitwise xor	`5 bxor 3 ⇒ 6`
	bsl	Bitshift left	`10 bsl 1 ⇒ 20`
	bsr	Bitshift right	`10 bsr 1 ⇒ 5`
	or	Logical or	`true or false ⇒ true`
	xor	Logical xor	`true xor false ⇒ true`
4	++	List concatenation	`[1,2]++[3] ⇒ [1,2,3]`
	--	List subtraction	`[1,2]--[2] ⇒ [1]`
5	==	Equal	`4 == 4.0 ⇒ true`
	/=	Not equal	`4 /= 4.0 ⇒ false`
	=<	Less or equal	`4 =< 5 ⇒ true`
	<	Less than	`4 < 5 ⇒ true`
	>=	Greater or equal	`4 >= 5 ⇒ false`
	>	Greater than	`4 > 5 ⇒ false`
	=:=	Exactly equal	`4 =:= 4.0 ⇒ false`
	=/=	Exactly not equal	`4 =/= 4.0 ⇒ true`
6	**andalso**	Short circuit and	`true andalso true ⇒ true`
7	**orelse**	Short circuit or	`true orelse false ⇒ true`
8	=	Match	`X = [1,2,3] ⇒ [1,2,3]`
	!	Send message	`Pid ! {1,2} ⇒ {1,2}`

Command Interface Functions

Function	*Description*
`c(File)`	Compile
`cd(Dir)`	Change directory
`f()`	Forget variable bindings
`ls()`	List directory
`pwd()`	Print working directory
`q()`	Quit

Guard Predicates

```
is_atom(X)         is_list(X)
is_binary(X)       is_number(X)
is_float(X)        is_pid(X)
is_function(X)     is_port(X)
is_integer(X)      is_tuple(X)
```

Guard Built-In Functions

```
abs(Number)
bit_size(Binary)
byte_size(Binary)
element(Index,Tuple)
float(Number)
hd(List)
length(List)
round(Number)
self()
tl(List)
trunc(Number)
tuple_size(Tuple)
```

`io:format(FormatString,DataList)`

Print to standard output items in `DataList`. `FormatString` may include: `~n` (newline), `~w` (write with standard syntax), `~s` (string), `~p` (pretty-print).

Spawn Functions

```
spawn(Fun)
spawn(Module,FunName,Args)
```

List Functions

```
lists:all(Pred,List)
lists:any(Pred,List)
lists:duplicate(N,X)
lists:filter(Pred,List)
lists:flatten(List)
lists:foldl(Fun,Acc,List)
lists:foldr(Fun,Acc,List)
lists:foreach(Fun,List)
lists:map(Fun,List)
lists:max(List)
lists:member(X,List)
lists:min(List)
lists:nth(Index,List)
lists:partition(Pred,List)
lists:reverse(List)
lists:seq(From,To)
lists:seq(From,To,Incr)
lists:sort(List)
lists:sum(List)
```

Parallel Lists Functions
http://code.google.com/p/plists/

```
plists:all(Pred,List)
plists:any(Pred,List)
plists:filter(Pred,List)
plists:fold(Pred,Acc,List)
plists:foreach(Fun,List)
plists:map(Fun,List)
plists:mapreduce(MapFunc,List)
plists:partition(Pred,List)
plists:sort(List)
```