Erlang Solutions Ltd.

Comprehensions

Comprehensions

- List Comprehensions
- Binary Comprehensions

List Comprehensions

```
[Expression || Generator1 , ..., GeneratorN,
               Filter1, ..., FilterN]
```

- A feature common in functional programming languages
- Analogous to set comprehensions in the Zermelo-Frankel set theory
- A syntactical and semantical notation to generate lists

List Comprehensions

```
[X || X <-[1,2,3,4], X < 3]
```

- The above example should be read as the list of X where X comes from the list [1,2,3,4] and X is less than 3
- **Pattern <- List** is the generator
- Filter is either a boolean expression or a function which returns true or false

List Comprehensions: examples

```
1> [Int || Int <- [zero, 1, two, three, 4, 5],
1>         is_integer(Int)].
[1,4,5]
2> [{X,Y} || X <- [1,2,3], Y <- [a,b,c]].
[{1,a},{1,b},{1,c},{2,a},{2,b},{2,c},{3,a},{3,b},{3,c}]
3> [X || X <- [1,2,3,4], Y <- [3,4,5,6], X == Y].
[3,4]
4> [X+1 || X <- [1,2,3,4], X rem 2 == 0].
[3,5]
```

- Filtering, cartesian products, intersections, and selective mapping using list comprehensions

List Comprehensions: examples

```
map(Fun, List) ->
    [Fun(X) || X <- List].
filter(Predicate, List) ->
    [X || X <- List, Predicate(X)].
append(ListOfLists) ->
    [X || List <- ListOfLists, X <- List].
```

- Rewriting lists library functions using list comprehensions

List Comprehensions: examples

```
perm([])  ->
    [[]];
perm(List) ->
    [[H|T] || H <- List, T <- perm(List -- [H])].
```

- perm([c,a,t]) -> [[c,a,t],[c,t,a],[a,c,t],[a,t,c],[t,c,a],[t,a,c]]
- We take **H** from **List** in all possible ways, and append all permutations of **List** with **H** removed to it

List Comprehensions: variables

- All variables in the generator pattern are considered fresh
- Bound variables in the generator and before the LC expression which are used in the filter retain their value
- No variable can be exported from a LC expression
- The compiler gives a warning when you shadow variables

List Comprehensions: variables

```
1> X = 1, Y = 2.
2
2> [{X, Y} || X <- lists:seq(1,3)].
[{1,2},{2,2},{3,2}]
3> List = [{1,one}, {2,two}, {3,three}].
[{1,one},{2,two},{3,three}]
4> [Z || {X1, Z} <- List, X1 == X].
[one]
```


Binary comprehensions

```
[ ... || X <- List, Test, ... ]
<< <<...>> || <<X>> <= Bin, Test, ... >>
```

- Structure similar to list comprehensions
- **Bin** must be a binary rather than a list
- A list generator can be used in a binary comprehension and a binary generator can be used in a list comprehension

Binary comprehensions: examples

```
1> << <<X>> || <<X>> <= <<0,1,2,3>> >>.
<<0,1,2,3>>
2> << <<(X+1)/integer>> || <<X>> <= <<3,7,5,4,7>> >>.
<<4,8,6,5,8>>
3> << <<(bnot X)>> || <<X>> <= <<0,1,2,3>> >>.
<<"ÿþýü">>
4> << <<X>> || X <- [0,1,2,3] >>.
<<0,1,2,3>>
5> [ X || <<X>> <= <<0,1,2,3>>].
[0,1,2,3]
6> << <<X>> || <<X>> <= <<1,2,3,4>>, X rem 2 =:= 0 >>.
<<2,4>>
```