BASIC ELEMENTS OF HASKELL BY EXAMPLE

WIM VANDERBAUWHEDE

We introduce some basic elements of Haskell through comparison with other languages

Expressions. In almost all programming languages you can create *expressions* such as:

```
(b*b-4*a*c)/2*a
```

and you can assign these expressions to variables:

```
v = (b*b-4*a*c)/2*a
```

In `Haskell`, you can do this as well, and what's more: expressions are really all there is, there are no statements.

Functions. In `Python`, you can define a function such as

```
def hello(name):
    return "Hello, "+name
```

In `Haskell` you can write this simply as:

```
hello name = "Hello, "++name
```

Types. `C` has *types*, for example:

```
int f (int x, int y) {
    return x*y+x+y;
}
```

`Haskell` has much more powerful types than `C`, and we will talk a lot about types:

```
f :: Int -> Int -> Int
f x y =  x*y+x+y
```

Lists. In many languages, e.g. Python, JavaScript, Ruby, ... you can create *lists* such as:

```
lst = [ "A", "list", "of", "strings"]
```

`Haskell` also uses this syntax for lists.

To join lists, in `Python` you could write

```
lst = [1,2] + [3,4]
```

In ‘Haskell this would be very similar:

```
lst = [1,2] ++ [3,4]
```

Anonymous functions. In `JavaScript` you can define *anonymous* functions (functions without a name) such as:

```
var f = function(x,y){return x*y+x+y};
```

In `Haskell`, such anonymous functions are called *lambda* functions and they are actually the basis of the language. Again, the syntax is very compact:

```
f = \x y -> x*y+x+y
```

Higher-order functions. Finally, in many languages, functions can operate on functions. For example, in `Perl` you can modify the elements in a list using:

```
map sub ($x){$x*2+1}, [1..10]
```

`Haskell` provides many of these so-called *higher-order functions*, and lets you define your own.

```
map (\x -> x*2+1) [1..10]
```