Informatics 1

Functional Programming Lecture 3

Thursday 18 September 2014

Lists and Comprehensions

Don Sannella

University of Edinburgh

Part I

List Comprehensions

Lists — Some examples

```
someNumbers :: [Integer]
someNumbers = [1,2,3]

someChars :: [Char]
   -- equivalent: someChars :: String
someChars = ['I','n','f','1']
   -- equivalent: someChars = "Inf1"

someLists :: [[Integer]]
someLists = [[1],[2,4,2],[],[3,5]]

someFunctions :: [Picture -> Picture]
someFunctions = [invert,flipV]

someStuff = [1,"Inf1",[2,3]]          -- type error!

someMoreNumbers :: [Integer]
someMoreNumbers = [1..10]
```

List comprehensions — Generators

```
Prelude> [ x*x | x <- [1,2,3] ]
[1,4,9]

Prelude> [ toLower c | c <- "Hello, World!" ]
"hello, world!"

Prelude> [ (x, even x) | x <- [1,2,3] ]
[(1,False),(2,True),(3,False)]
```

`x <- [1,2,3]` is called a *generator*

`<-` is pronounced *drawn from*

List comprehensions — Guards

```
Prelude> [ x | x <- [1,2,3], odd x ]
[1,3]

Prelude> [ x*x | x <- [1,2,3], odd x ]
[1,9]

Prelude> [ x | x <- [42,-5,24,0,-3], x > 0 ]
[42,24]

Prelude> [ toLower c | c <- "Hello, World!", isAlpha c ]
"helloworld"
```

`odd x` is called a *guard*

Sum, Product

```
Prelude> sum [1,2,3]
6

Prelude> sum []
0

Prelude> sum [ x*x | x <- [1,2,3], odd x ]
10

Prelude> product [1,2,3,4]
24

Prelude> product []
1

Prelude> let factorial n = product [1..n]
Prelude> factorial 4
24
```

Example uses of comprehensions

```
squares :: [Integer] -> [Integer]
squares xs  =  [ x*x | x <- xs ]

odds :: [Integer] -> [Integer]
odds xs  =  [ x | x <- xs, odd x ]

sumSqOdd :: [Integer] -> Integer
sumSqOdd xs  =  sum [ x*x | x <- xs, odd x ]
```

QuickCheck

```
-- sumSqOdd.hs

import Test.QuickCheck

squares :: [Integer] -> [Integer]
squares xs  =  [ x*x | x <- xs ]

odds :: [Integer] -> [Integer]
odds xs  =  [ x | x <- xs, odd x ]

sumSqOdd :: [Integer] -> Integer
sumSqOdd xs  =  sum [ x*x | x <- xs, odd x ]

prop_sumSqOdd :: [Integer] -> Bool
prop_sumSqOdd xs  =  sum (squares (odds xs)) == sumSqOdd xs
```

Running QuickCheck

```
[melchior]dts: ghci sumSqOdd.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( sumSqOdd.hs, interpreted )
*Main> quickCheck prop_sumSqOdd
Loading package old-locale-1.0.0.0 ... linking ... done.
Loading package old-time-1.0.0.0 ... linking ... done.
Loading package random-1.0.0.0 ... linking ... done.
Loading package mtl-1.1.0.1 ... linking ... done.
Loading package QuickCheck-2.1 ... linking ... done.
+++ OK, passed 100 tests.
*Main>
```