Lesson 3: List Comprehensions

What's It All About?

1. Introduce the form and function of List Comprehensions
2. Example Numeric Comprehensions
3. Example String Comprehensions

What are List Comprehensions?

In mathematics, a "set comprehension" might look something like this:

$\{ \; x^2 \mid \mathrm{x} \in \mathrm{N}, \; \mathrm{x} \leq 10 \; \}$

What does this mean? It means this:

1. Generate the set of all natural numbers which are less than 10.
2. Produce a new set consisting of the squares of all of the generated numbers.

Thus, the example set comprehension would internally generate
{1,2,3,4,5,6,7,8,9,10},
and then transform these numbers to their squares and produce
{1,4,9,16,25,36,49,64,81,100}.

In Haskell, "list comprehensions", which are modelled after set comprehensions in mathematics, are a very popular way to generate, combine, transform and filter lists. For example, the following list comprehension produces the list of the squares of the first 10 natural numbers:

```
>>> [ x*x | x <- [1..10] ]
[1,4,9,16,25,36,49,64,81,100]
```

Here is an example of a list comprehension with a filter:

```
>>> [ x*x | x <- [1..10], even x ]
[4,16,36,64,100]
```

List comprehensions can feature Strings, since character strings are represented as lists of characters in Haskell. For example:

```
>>> [ s | s <- ["red","yellow","blue","purple"]]
["red","yellow","blue","purple"]
>>> [ s | s <- ["red","yellow","blue","purple"], length s < 5]
["red","blue"]
>>>
```

Moreover list comprehensions can feature multiple variables, as shown in the following two more examples, which are somewhat reminiscent of truth tables:

```
>>> [ (a,b) | a <- [True,False], b <- [True,False] ]
[(True,True),(True,False),(False,True),(False,False)]
>>> [ [a,b,c] | a <- "TF", b <- "TF", c <- "TF" ]
["TTT","TTF","TFT","TFF","FTT","FTF","FFT","FFF"]
>>>
```

Cartesian Coordinate Variations

```
>>> [ (x,y) | x <- [1,2,3], y <- [1..3] ]
[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]
>>> [ (x,y) | x <- [1,2,3], y <- [x..3] ]
[(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)]
>>> [ (x,y) | x <- [1,2,3], y <- [1..4], x+y == 5]
[(1,4),(2,3),(3,2)]
>>>
```

Sequences of Multiples of a Number

```
>>> [ a | a <- [1..10] ]
[1,2,3,4,5,6,7,8,9,10]
>>> :set prompt ">>> "
>>> [ a*3 | a <- [1..10] ]
[3,6,9,12,15,18,21,24,27,30]
>>> [ a*10 | a <- [1,2,3] ]
[10,20,30]
>>> multiples n m = [ a*m | a <- [1..n] ]
>>> multiples 15 3
[3,6,9,12,15,18,21,24,27,30,33,36,39,42,45]
>>> multiples 6 7
[7,14,21,28,35,42]
>>>
```

Functions with List Comprehensions

Code

```
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
--- fourth_functions.hs contains some list comprehensions

-----------------------------------------------------------------------------
-- Thing 1
```

```
factors n = [ x | x <- [1..n], n ‘mod‘ x == 0 ]

prime n = factors n == [1,n]

primes n = [ x | x <- [2..n], prime x ]

------------------------------------------------------------------------------
-- Thing 2

p3 w = [ [a,b,c] | a <- w, b <- w, c <- w, distinct [a,b,c]]

p4 w = [ [a,b,c,d] | a <- w, b <- w, c <- w, d <-w, distinct [a,b,c,d]]

p5 w = [ [a,b,c,d,e] | a <- w, b <- w, c <- w, d <- w,  e <- w, distinct [a,b,c,d,e] ]

distinct [] = True
distinct (x:xs) = if ( elem x xs ) then False else distinct xs

------------------------------------------------------------------------------
-- Thing 3

laughter digits = [ if ( odd d ) then "HA" else "HE" | d <- digits ]

------------------------------------------------------------------------------
-- Thing 4

inps = [ a ++ " " ++ q ++ " " ++ n |
         a <- articles, q <- adjectives, n <- nouns, match a q
       ]
   where articles = ["a","an"]
         adjectives = ["orange","blue","wierd","elegant"]
         nouns = ["cow","computer","coffee cup"]
         match "a" (f:_) = elem f "bcdfghjklmnpqrstvwxyz"
         match "an" (f:_) = elem f "aeiou"
```

Demo

```
bash-3.2$ ghci
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "

>>> :load fourth_functions
[1 of 1] Compiling Main             ( fourth_functions.hs, interpreted )
Ok, one module loaded.

>>> factors 24
[1,2,3,4,6,8,12,24]
>>> factors 19
[1,19]
>>> :type factors
factors :: Integral a => a -> [a]
```

```
>>> prime 24
False
>>> prime 19
True
>>> :type prime
prime :: Integral a => a -> Bool

>>> primes 24
[2,3,5,7,11,13,17,19,23]
>>> primes 50
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]
>>> :type primes
primes :: Integral a => a -> [a]
>>>

>>> p3 [1,2,3]
[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
>>> p3 "cat"
["cat","cta","act","atc","tca","tac"]
>>> :type p3
p3 :: Eq a => [a] -> [[a]]

>>> p4 [1,2,3,4]
[[1,2,3,4],[1,2,4,3],[1,3,2,4],[1,3,4,2],[1,4,2,3],[1,4,3,2],[2,1,3,4],
[2,1,4,3],[2,3,1,4],[2,3,4,1],[2,4,1,3],[2,4,3,1],[3,1,2,4],[3,1,4,2],
[3,2,1,4],[3,2,4,1],[3,4,1,2],[3,4,2,1],[4,1,2,3],[4,1,3,2],[4,2,1,3],
[4,2,3,1],[4,3,1,2],[4,3,2,1]]
>>> p4 "blue"
["blue","bleu","bule","buel","belu","beul","lbue","lbeu","lube","lueb",
"lebu","leub","uble","ubel","ulbe","uleb","uebl","uelb","eblu","ebul",
"elbu","elub","eubl","eulb"]
>>> :type p4
p4 :: Eq a => [a] -> [[a]]

>>> p5 "cigar"
["cigar","cigra","ciagr","ciarg","cirga","cirag","cgiar","cgira","cgair",
"cgari","cgria","cgrai","caigr","cairg","cagir","cagri","carig","cargi",
"criga","criag","crgia","crgai","craig","cragi","icgar","icgra","icagr",
"icarg","icrga","icrag","igcar","igcra","igacr","igarc","igrca","igrac",
"iacgr","iacrg","iagcr","iagrc","iarcg","iargc","ircga","ircag","irgca",
"irgac","iracg","iragc","gciar","gcira","gcair","gcari","gcria","gcrai",
"gicar","gicra","giacr","giarc","girca","girac","gacir","gacri","gaicr",
"gairc","garci","garic","grcia","grcai","grica","griac","graci","graic",
"acigr","acirg","acgir","acgri","acrig","acrgi","aicgr","aicrg","aigcr",
"aigrc","aircg","airgc","agcir","agcri","agicr","agirc","agrci","agric",
"arcig","arcgi","aricg","arigc","argci","argic","rciga","rciag","rcgia",
"rcgai","rcaig","rcagi","ricga","ricag","rigca","rigac","riacg","riagc",
"rgcia","rgcai","rgica","rgiac","rgaci","rgaic","racig","racgi","raicg",
"raigc","ragci","ragic"]
>>> :type p5
p5 :: Eq a => [a] -> [[a]]

>>> laughter [1,2,3,4,5]
```

```
["HA","HE","HA","HE","HA"]
>>> laughter [1,1,2,2]
["HA","HA","HE","HE"]

>>> inps
["a blue cow","a blue computer","a blue coffee cup","a wierd cow",
"a wierd computer","a wierd coffee cup","an orange cow",
"an orange computer","an orange coffee cup","an elegant cow",
"an elegant computer","an elegant coffee cup"]

>>>
```