Tutorial 4A: List comprehension

A list comprehension is a convenient notation for list operations. The syntax is designed after the way sets are described in mathematics, e.g. the set of all primes:

$$\{n \mid n \in \mathbb{N}, n \text{ is prime}\}$$

For simplicity we take the predicate "is prime" as given. In Haskell, if we are given a function `prime :: Int -> Bool` that tells us whether a given number is prime, we can get the list of all prime numbers by a list comprehension as follows:

```
primes :: [Int]
primes = [ n | n <- [1..] , prime n ]
```

This gives the list of integers `n` that are **drawn from** the infinite list `[1..]` and that satisfy the condition `prime n`. Here, we collect just the numbers `n`, but any function over `n` is permitted. A list comprehension is constructed as follows:

```
[ <exp> | <qualifier_1> , ... , <qualifier_n> ]
```

where each qualifier can be a **generator** `x <- xs` that **draws** elements `x` from a list `xs` (such as `n <- [1..]` above), or a boolean **guard** (such as `prime n` above). List comprehensions are intimately related to maps and filters:

```
map f (filter p xs)   ⇔   [ f x | x <- xs , p x ]
```

Exercise 1: Write the following functions from Tutorial 3A again, this time using a list comprehension:

a) `doubles` , which multiplies every number in a list by two,

b) `odds` , which removes any even numbers from a list,

c) `doubleodds` , which doubles every odd number in a list (removing the even ones),

d) `shorts` , which removes all strings longer than 5 characters from a list,

e) `squarePositives` , which squares all positive integers in a list (and removes the negative ones and zero),

f) `oddLengthSums` , which for a list of integer lists returns the sum of each odd-length list,

g) `remove` , which removes all occurrences of an element from a list,

h) `removeAll` , which removes the elements of the second list from the first,

i) `everyother` , which takes every other element from a list starting with the first (use `zip` again),

j) `same` , which takes two lists and returns a list of the positions where their elements coincide; you may use `zip` and `zipWith` again, or instead try it with

```
zip3 :: [a] -> [b] -> [c] -> [(a,b,c)] .
```

```
*Main> doubles [1..10]
[2,4,6,8,10,12,14,16,18,20]
*Main> odds [1..10]
[1,3,5,7,9]
*Main> doubleodds [1..10]
[2,6,10,14,18]
*Main> shorts ["The","truth","is","that","in","London","it","is",
                                    "always","a","sickly","season"]
["The","truth","is","that","in","it","is","a"]
*Main> squarePositives [-3,4,1,-2,0,3]
[16,1,9]
*Main> oddLengthSums [[1],[1,2],[1,2,3],[1..4],[1..5]]
[1,6,15]
*Main> remove "George" 'e'
"Gorg"
```

```
*Main> removeAll "Fitzwilliam" "Fitz"
"wllam"
*Main> everyother "Elizabeth"
"Eiaeh"
*Main> same "Charles" "Charlotte"
[1,2,3,4,5]
```

Exercise 2:

a) Complete the function `pairs` which takes two lists and produces every possible combination of items. Use a list comprehension with two generators. (**Bonus:** adapt this to a function `applies :: [a->b] -> [a] -> [b]` that takes a list of functions and a list of arguments and applies each function to each argument.)

b) Complete the function `selfpairs` which takes a list and produces every possible pairing of its items, avoiding symmetric duplicates. That is, for each element, pair it only with those coming **after** it in the list, and with itself. In your list comprehension, use `zip` to count elements and `drop` to obtain those after that index.

c) Complete the function `pyts` so that `pyts n` generates all ordered Pythagorean triples with numbers up to (and including) `n`.

```
*Main>  pairs "xy" [1,2,3]
[('x',1),('x',2),('x',3),('y',1),('y',2),('y',3)]
*Main> selfpairs [1..4]
[(1,1),(1,2),(1,3),(1,4),(2,2),(2,3),(2,4),(3,3),(3,4),(4,4)]
*Main> pyts 100
[(3,4,5),(5,12,13),(6,8,10),(7,24,25),(8,15,17),(9,12,15),
(9,40,41),(10,24,26),(11,60,61),(12,16,20),(12,35,37),(13,84,85),
(14,48,50),(15,20,25),(15,36,39),(16,30,34),(16,63,65),(18,24,30),
(18,80,82),(20,21,29),(20,48,52),(21,28,35),(21,72,75),(24,32,40),
(24,45,51),(24,70,74),(25,60,65),(27,36,45),(28,45,53),(28,96,100),
(30,40,50),(30,72,78),(32,60,68),(33,44,55),(33,56,65),(35,84,91),
(36,48,60),(36,77,85),(39,52,65),(39,80,89),(40,42,58),(40,75,85),
(42,56,70),(45,60,75),(48,55,73),(48,64,80),(51,68,85),(54,72,90),
(57,76,95),(60,63,87),(60,80,100),(65,72,97)]
```