Chapter 5 Exercises

List Comprehensions

Connor Baker

Compiled on December 21, 2019 at 9:03pm

5.7 Exercises

1. Using a list comprehension, give an expression that calculates the sum $1^2 + 2^2 + ... + 100^2$ of the first one hundred integer squares.

   Solution:

```
sum100ConsecSquares :: Int
sum100ConsecSquares = sum [x^2 | x <- [1..100]]
```

2. Suppose that a coordinate grid of size $m \times n$ is given by the list of all pairs (x, y) of integers such that $0 \leq x \leq m$ and $0 \leq y \leq n$. Using a list comprehension, define a function `grid :: Int -> Int -> [(Int,Int)]` that returns a coordinate grid of a given size. For example:

```
> grid 1 2
[(0,0),(0,1),(0,2),(1,0),(1,1),(1,2)]
```

   Solution:

```
grid :: Int -> Int -> [(Int,Int)]
grid m n = [(x,y) | x <- [0..m], y <- [0..n]]
```

3. Using a list comprehension and the function grid above, define a function `square :: Int -> [(Int,Int)]` that returns a coordinate square of size n, excluding the diagonal from $(0, 0)$ to (n, n). For example:

```
> square 2
[(0,1),(0,2),(1,0),(1,2),(2,0),(2,1)]
```

   Solution:

```
grid :: Int -> Int -> [(Int,Int)]
grid m n = [(x,y) | x <- [0..m], y <- [0..n]]

square :: Int -> [(Int,Int)]
square n = [(x,y) | (x,y) <- grid n n, x /= y]
```

4. In a similar way to the function length, show how the library function `replicate :: Int -> a -> [a]` that produces a list of identical elements can be defined using a list comprehension. For example:

```
> replicate 3 True
[True,True,True]
```

   Solution:

```
replicate :: Int -> a -> [a]
replicate n x = [x | _ <- [1..n]]
```

5. A triple (x, y, z) of positive integers is Pythagorean if it satisfies the equation $x^2 + y^2 = z^2$. Using a list comprehension with three generators, define a function `pyths :: Int -> [(Int,Int,Int)]` that returns the list of all such triples whose components are at most a given limit. For example:

```
> pyths 10
[(3,4,5),(4,3,5),(6,8,10),(8,6,10)]
```

By adding in the constraints that $a \leq b \leq c$ we can uniquely generated pythagorean triplets which are unique up to rotation.

Solution:

```
pyths :: Int -> [(Int,Int,Int)]
pyths n = [(a,b,c) | c <- [1..n],
                     b <- [1..c],
                     a <- [1..b],
                     a^2 + b^2 == c^2]
```

6. A positive integer is perfect if it equals the sum of all of its factors, excluding the number itself. Using a list comprehension and the function factors, define a function `perfects :: Int -> [Int]` that returns the list of all perfect numbers up to a given limit. For example:

```
> perfects 500
[6,28,496]
```

Solution:

```
factors :: Int -> [Int]
factors n = [x | x <- [1..n], n `mod` x == 0]

perfects :: Int -> [Int]
perfects n = [x | x <- [1..n],
                x == sum (factors x) - ]
```

7. Show how the list comprehension `[(x,y)| x <- [1,2], y <- [3,4]]` with two generators can be re-expressed using two comprehensions with single generators. Hint: nest one comprehension within the other and make use of the library function `concat :: [[a]] -> [a]`.

Solution:

```
list1 :: [(Int,Int)]
list1 =  concat [[(x,y) | y <- [3,4]] | x <- [1,2]]
```

8. Redefine the function positions using the function find.

Solution:

```
find :: Eq a => a -> [(a,b)] -> [b]
find k t = [v | (k',v) <- t, k == k']

positions :: Eq a => a -> [a] -> [Int]
positions x xs = [i | (x',i) <- zip xs [0..], x == x']

positions' :: Eq a => a -> [a] -> [Int]
positions' x xs = find x (zip xs [0..])
```

9. The scalar product of two lists of integers `xs` and `ys` of length n is given by the sum of the products of corresponding integers:

$$\sum_{i=0}^{n-1} xs_i \cdot ys_i$$

In a similar manner to `chisqr`, show how a list comprehension can be used to define a function `scalarproduct :: [Int] -> [Int] -> Int` that returns the scalar product of two lists. For example:

```
> scalarproduct [1,2,3] [4,5,6]
32
```

Solution:

```
chisqr :: [Float] -> [Float] -> Float
chisqr os es = sum [((o-e)^2)/e | (o,e) <- zip os es]

scalarproduct :: [Int] -> [Int] -> Int
scalarproduct xs ys = sum [x*y | (x,y) <- zip xs ys]
```

10. Modify the Caesar cipher program to also handle upper-case letters.

Solution:

```
-- We assume that the input is not only upper case letters.
import Data.Char

-- Returns a list of indices of the specifed value
positions :: Eq a => a -> [a] -> [Int]
positions x xs = [i | (x',i) <- zip xs [0..], x == x']

-- Returns the number of lowercase characters
lowers :: String -> Int
lowers xs = length [x | x <- xs, 'a' <= x && x <= 'z']

-- Returns the number of occurences
count :: Char -> String -> Int
count x xs = length [x' | x' <- xs, x == x']

-- Converts a character into an integral value
-- Specifically, it maps [a..z]++[A..Z] to the interval [0..51]
let2int :: Char -> Int
let2int c | 'a' <= c && c <= 'z' = ord c - ord 'a'
          | 'A' <= c && c <= 'Z' = ord c - ord 'A' + 26
          | otherwise            = ord c

-- Converts an integral value into a character
int2let :: Int -> Char
int2let n | n <= 25 = chr (ord 'a' + n)
          | n <= 51 = chr (ord 'A' - 26 + n)
          | otherwise = chr n
```

```haskell
shift :: Int -> Char -> Char
shift n c
    | isLower c = int2let ((let2int c + n) `mod` 26)
    | isUpper c = int2let (((let2int c + n - 26) `mod` 26) + 26)
    | otherwise = c

encode :: Int -> String -> String
encode n xs = [shift n x | x <- xs]

-- Frequency distribution of lowercase english letters
table :: [Float]
table = [8.1, 1.5, 2.8, 4.2, 12.7, 2.2, 2.0, 6.1, 7.0,
        0.2, 0.8, 4.0, 2.4, 6.7, 7.5, 1.9, 0.1, 6.0,
        6.3, 9.0, 2.8, 1.0, 2.4, 0.2, 2.0, 0.1]

percent :: Int -> Int -> Float
percent n m = (fromIntegral n / fromIntegral m) * 100

-- Only necessary to count the number of lowercase letters present.
freqs :: String -> [Float]
freqs xs = [percent (count x xs) n | x <- ['a'..'z']]
    where n = lowers xs

chisqr :: [Float] -> [Float] -> Float
chisqr os es = sum [((o-e)^2)/e | (o,e) <- zip os es]

rotate :: Int -> [a] -> [a]
rotate n xs = drop n xs ++ take n xs

crack :: String -> String
crack xs = encode (-factor) xs
    where
        factor = head (positions (minimum chitab) chitab)
        chitab = [chisqr (rotate n table') table | n <- [0..25]]
        table' = freqs xs
```