Worksheet 2: *Lists*

Template file:	Worksheet2.hs
Labs:	Friday 8th February, 2019
Hand-in:	Sunday 17st February, 2019at 18:00hr
Topics:	Lists. Map and filter. List comprehension.

Message: (1) don't forget to register your attendance, (2) don't forget to put your name on your script, (3) scripts that don't compile properly may loose 20% points, (4) don't try to find solutions on the web, you learn by doing it yourself! The answers are generally short. The convenor can ask you to explain your code. (5) Take care that the layout is pleasing. (6) More importantly have fun!

1. An phone book for storing names and telephone numbers can be implemented in Haskell as follows.

```
type Name = String
type PhoneNumber = Int
type Person  = (Name, PhoneNumber)
type PhoneBook = [Person]
```

   (a) (5 marks) Write the function `add::Person -> PhoneBook -> PhoneBook` that adds an entry to the phone book at the beginning of the list.

   (b) (5 marks) Write the function `delete::Name -> PhoneBook -> PhoneBook` that given the name of a person deletes all entries in the phone book with that name.

   (c) (5 marks) Write the function `find::Name -> PhoneBook -> [PhoneNumber]` that gives the list of all telephone numbers of a certain person.

   (d) (5 marks) Write the function `update::Name -> PhoneNumber -> PhoneNumber -> PhoneBook -> PhoneBook` that given the name and old phone number of a person updates that entry in the phone book with that the new phone number. (You may assume that the phonebook does not contain multiple entries of the same data.)

2. A Bank stores details on its customers via their national insurance number, their age, and their balance. This gives the following type definitions.

```
type NI = Int
type Age = Int
type Balance = Float
type Customer  = (NI,Age, Balance)
type Bank = [Customer]
```

   (a) (5 marks) Define a function `retired :: Customer -> Bool` which returns true if the person is, or is over, 67 years.

(b) (5 marks) Define a function `deposit :: Customer -> Float -> Customer` which adds a given amount to the person's balance.

(c) (5 marks) Define a function `withdraw :: Customer -> Float -> Customer` which removes a given amount from the person's balance, but only if the remaining total is positive!

(d) (5 marks) Define a function `credit :: Bank -> [Customer]` which returns those people who are not overdrawn.

3. (a) (5 marks) Using list comprehension define the function `cubeOdds :: [Int] -> [Int]` which takes a list of integers as input and returns a list consisting of the cube of only the odd numbers, eg `cubeOdds [3,6,4,5] = [27,125]`.

   (b) (5 marks) Define also a function `cubeOdds2` which has the same effect as `cubeOdds` but which is defined using map and filter instead of list comprehension.

4. (Challenge, not assessed) Define a function `addIndex :: [Int] -> [(Int,Int)]` that given a list $[n_1, n_2, \ldots n_k]$ of integers produces the list $[(1, n_1), (2, n_2), \ldots, (k, n_k)]$ which is a list of pairs of integers.

   For example `addIndex [2,2,3,1] -> [(1,2),(2,2),(3,3),(4,1)]`