Declarative Programming

Workshop exercises set 5.

QUESTION 1
Define the function

```
maybeApply :: (a -> b) -> Maybe a -> Maybe b
```

that yields Nothing when the input Maybe is Nothing, and
applies the supplied function to the content of the Maybe when it is
Just some content.

Try, for example, computing

```
maybeApply (+1) (Just 41)
maybeApply (+1) Nothing
```

This function is defined in the standard prelude as fmap.

QUESTION 2
Define the function

```
zWith :: (a -> b -> c) -> [a] -> [b] -> [c]
```

that constructs a list of the result of applying the first argument to
corresponding elements of the two input lists. If the two list
arguments are different lengths, the extra elements of the longer one
are ignored. For example,

```
zWith (-) [1,4,9,16] [1,2,3,4,5] = [0,2,6,12]
```

This function is defined in the standard library as 'zipWith'.

QUESTION 3
Define the function

```
linearEqn :: Num a => a -> a -> [a] -> [a]
```

that constructs a list of the result of multiplying each element in the
third argument by the first argument, and then adding the second argument.
For example,

```
linearEqn 2 1 [1,2,3] = [2*1+1, 2*2+1, 2*3+1] = [3,5,7]
```

Write the simplest defintion you can, remembering the material covered
recently.

QUESTION 4
The following function takes a number and returns a list containing the
positive and negative square roots of the input (assume non-zero input)

```
>sqrtPM :: (Floating a, Ord a) => a -> [a]
>sqrtPM x
>  | x  > 0    = let y = sqrt x in [y, -y]
>  | x == 0    = [0]
>  | otherwise = []
```

Using this function, define a function allSqrts that takes a list and
returns a list of all the positive and negative square roots of all
the numbers on the list. For example:

```
    allSqrts [1,4,9] = [1.0,-1.0,2.0,-2.0,3.0,-3.0]
```

Include a type declaration for your function.

QUESTION 5
Lectures have given the definitions of two higher order functions in the Haskell prelude, filter and map:

```
        filter :: (a -> Bool) -> [a] -> [a]
        map :: (a -> b) -> [a] -> [b]
```

Filter returns those elements of its argument list for which the given function returns True, while map applies the given function to every element of the given list.

Suppose you have a list of numbers, and you want to (a) filter out all the negative numbers, and (b) apply the sqrt function to all the remaining integers.

(a) Write code to accomplish this task using filter and map.
(b) Write code to accomplish this task that does only one list traversal, without any higher order functions.
(c) Transform (b) to (a).