IN3043 Functional Programming
Exercises 3

1. In the interpreter, write expressions for

   (a) All the numbers from 1 to 100.

   (b) Squares of the numbers from 1 to 20.

   (c) Divisors of 100. (*Hint*: these will be numbers `n` between 1 and 100 such that `100 ‘mod‘ n == 0`.)

2. This function triples each integer in a list:

   ```
   tripleAll :: [Int] -> [Int]
   tripleAll ns = [3*n | n <- ns]
   ```

   In your source file, give a definition of the similar function

   ```
   squareAll :: [Int] -> [Int]
   ```

   that squares all the elements of a list of integers.

3. Give a definition of the function

   ```
   capitalize :: String -> String
   ```

   that returns the input list with the lower case letters capitalized and the others unchanged. You will need to use the function

   ```
   toUpper :: Char -> Char
   ```

   from the `Data.Char` module. (You'll need `import Data.Char` in your file.)

4. Generalizing the previous exercise, write a function

   ```
   capitalizeLetters :: String -> String
   ```

   that does the same, but discards non-letters. You may wish to use the following function from the `Data.Char` module:

   ```
   isAlpha :: Char -> Bool
   ```

5. Write a function

```
divisors :: Int -> [Int]
```

returning the list of numbers that evenly divide the given number.

6. What are the values of the following expressions? (The interpreter can work them out for you, but you need to get to the point where you can predict the answers.)

   (a) `[10*x+y | x <- [1..4], y <- [1..3]]`

   (b) `[10*x+y | x <- [1..4], y <- [x..2*x]]`

   (c) `[10*x+y | x <- [1..4], y <- [x..2*x], (x+y) ‘mod‘ 3 == 0]`

7. A *palindrome* is a word that is the same reversed, e.g. *kayak* or *repaper*. (This doesn't need a list comprehension.)

   (a) Write a function to test whether a word is a palindrome.

   (b) Some famous palindromes rely on ignoring non-letters and the distinction between upper and lower case (e.g. "Madam, I'm Adam"). Write a function to test for this. (Hint: can you use functions you've previously defined?)

8. Write a function

```
backwards :: String -> String
```

that takes a string consisting of words, and returns a string of the same words in the reverse order.

9. Write a function of the same type that takes a string consisting of words, and returns a string of the same words in the same order, but with each word reversed.

10. Define a function

```
capitalPositions :: String -> [Int]
```

that returns the positions of the capital letters (counting from 0) in the original list, e.g.

```
capitalPositions "Hello World" = [0,6]
```

11. Suppose an entry for the table of contents of a book is represented by a pair of a `String` (the chapter or section name) and an `Int` (the page number), e.g.

```
("5.4 Lists in Haskell", 79)
```

(a) Write a function to compute a contents line 40 characters wide from such a string, padding the gap with dots, e.g.

```
"5.4 Lists in Haskell ............... 79"
```

You can use the Prelude function

```
show :: Int -> String
```

to convert the page number into a string.

(b) Write a function

```
toc :: [(String,Int)] -> String
```

that formats a table of contents as a single string, with lines separated by `'\n'` characters. To print this output neatly, enter at the GHCi command line

```
putStr (toc entries)
```

where `entries` is the list of pairs. The function `putStr` produces a special value that tells the GHCi system to print the string literally (more on this in week 11).

12. The standard library contains a function

```
elem :: Eq a => a -> [a] -> Bool
```

Work out what this function does, and then define an equivalent function using library functions (other than `elem` itself) and list comprehensions.