Comp101P
Lab exercise sheet 1

Log into Linux, open a terminal window, create a suitable directory in your directory tree and change to this directory. Then start GHCi by entering the command `ghci` at the prompt. Open another terminal, change to the same directory and then open your favourite editor in the terminal if it is an in-terminal editor. Otherwise make sure it saves its files in the same directory in which you opened GHCi.

Simple functions, lists, tuples, list comprehension and types

- Use the editor to open and start writing a file called `LabSheet1.hs`
- Consider a function called `square` which squares integers.
- In the file `LabSheet1.hs` write a type and a definition for `square`. Save and then load this into your environment using the command
  `:l LabSheet1`
- Check that your function behaves correctly by testing your function with a range of examples.
- Consider a function called `pyth` which takes a pair of integers and returns the sum of the squares of the two integers.
- In the file `LabSheet1.hs` write a type and a definition for `pyth` so that it calles `square`. Save and then load this into your environment using the command
  `:r`
- Check that your function behaves correctly by testing it with a range of examples.
- Write (with a type declaration) a function `isTriple` that takes three integers and checks whether they form the sides of a right angled triangle. The last number should be the hypotenuse. Use the function `pyth`.

- Improve `isTriple` so that the hypotenuse can be in any position. Call the new function `isTripleAny`.
- Use the functions `div, mod :: Int -> Int -> Int` and list comprehension to write a function `halfEvens :: [ Int ] -> [ Int ]` which halves each even number in a list. E.g.

```
halfEvens [1,2,3,4,5,6] == [1,1,3,2,5,3]
```

- Use list comprehension to write a function `inRange :: Int -> Int -> [Int] -> [Int]` to return all numbers in the input list within the range given by the first two arguments (inclusive). For example,

```
inRange 5 10 [1..15] == [5,6,7,8,9,10]
```

- Write a function `countPositives` to count the positive numbers in a list (the ones strictly greater than 0). For example,

```
countPositives [0,1,-3,-2,8,-1,6] == 3
```

  Your definition should use a list comprehension and a list library function.
- Write a function `capitalised :: String -> String` which, given a word, capitalises it. That means that the first character should be made uppercase and any other letters should be made lowercase. For example,

```
capitalised "mELboURNe" == "Melbourne"
```

  Your definition should use a list comprehension and the library functions `toUpper` and `toLower` that change the case of a character. Use the internet to find out which library module they are in and how to load a module.
- Using the function `capitalised` from the previous question, write a function `title :: [String] -> [String]` which, given a list of words, capitalises them as a title should be capitalised. The proper capitalisation of a title (for our purposes) is as follows: The first word should be capitalised. Any other word should be capitalised if it is at least four letters long. For example,

```
title ["tHe", "bOSun", "ANd", "thE", "BriDGe"]
  == ["The", "Bosun", "and", "the", "Bridge"]
```

  Your function should use a list comprehension, and you will probably need some other auxiliary functions. You may use library functions that change the case of a character and the function `length`. You will need to write a recursive definition to cover the case when the string is empty.