Comprehensions
Informatics 1 – Introduction to Computation
Functional Programming Tutorial 2

Banks, Cooper, Fehrenbach, Heijltjes, Melkonian, Sannella, Vlassi-Pandi, Wadler

Week 3
due 12:00 Tuesday 3 October 2023
tutorials on Thursday 5 and Friday 6 October 2023

You will not receive credit for your coursework unless you attend the corresponding tutorial session. Please email kendal.reid@ed.ac.uk if you cannot join your assigned tutorial.

Good Scholarly Practice: Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct.

This also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.

1 List Comprehension

The present tutorial is about understanding *list comprehensions*.

Exercise 1

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]
```

Your definition should use a *list comprehension*.

Exercise 2

(a) Write a function `multDigits :: String -> Int` that returns the product of all the digits in the input string. If there are no digits, your function should return 1. For example,

```
multDigits "The time is 4:25"  ==  40
multDigits "No digits here!"   ==   1
```

Your definition should use a *list comprehension*. You'll need library functions to determine if a character is a digit, to convert a digit to an integer, and to find the product of a list.

(b) Write a function `countDigits :: String -> Int` that returns the number of digits in the input string. Your definition should use a *list comprehension* and a suitable library function.

(c) Because 9 is the largest digit, the number returned by `multDigits` on any given input should be less than or equal to 9^x where x is the number of digits as returned by `countDigits`. Write and execute a QuickCheck property `prop_multDigits` to confirm. The exponentiation operator is `(^)`, e.g. `9 ^ 3` $= 9^3 = 729$.

Exercise 3

(a) Write a function `capitalise :: 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,

```
capitalise "edINBurgH"  ==  "Edinburgh"
```

Your definition should use a *list comprehension*. You'll need library functions to change a character to upper and lower case, and you may want to write an auxiliary helper function.

Hint: Use pattern matching to decompose the input string into the first character and the rest.

(b) Using `capitalise`, 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", "sOunD", "ANd", "thE", "FuRY"]
==  ["The", "Sound", "and", "the", "Fury"]
```

Your function should use a *list comprehension*, and you may want to write an auxiliary helper function.

Exercise 4

(a) Write a function `score :: Char -> Int` that converts a character to its score. Each letter starts with a score of one; one is added to the score of a character if it is a vowel (a, e, i, o, u) and one is added to the score of a character if it is upper case; a character that is not a letter scores zero. For example,

```
score 'A' == 3
score 'a' == 2
score 'B' == 2
score 'b' == 1
score '.' == 0
```

(b) Write a function `totalScore :: String -> Int` that given a string returns the product of the score of every letter in the string, ignoring any character that is not a letter. For example,

```
totalScore "aBc4E" == 12
```

(c) Write a test function `prop_totalScore_pos` that checks that `totalScore` always returns a number greater than or equal to one.

2 Optional Material

Following the Common Marking Scheme, a student with good mastery of the material is expected to get 3/4 points. This section is for demonstrating exceptional mastery of the material. It is optional and worth 1/4 points.

In this optional part of the tutorial we will use *list comprehensions* to write some more involved functions.

Exercise 5

Dame Curious is a crossword enthusiast. She has a list of words that might appear in a crossword puzzle, but she has trouble finding the ones that fit a slot. Write a function

```
crosswordFind :: Char -> Int -> Int -> [String] -> [String]
```

to help her. The expression

```
crosswordFind letter pos len words
```

should return all the items from `words` which (a) are of the given length `len` and (b) have `letter` in position `pos`, starting counting with position 0. For example, if Curious is looking for seven-letter words that have `'k'` in position 1, she can evaluate the expression:

```
crosswordFind 'k' 1 7 ["funky", "fabulous", "kite", "icky", "ukelele"]
```

to get `["ukelele"]`.

Your definition should use a *list comprehension*. You may also use a library function which returns the nth element of a list, for a given argument n, and the function `length`.

Exercise 6

(a) Write a function `search :: String -> Char -> [Int]` that returns the positions of all occurrences of the second argument in the first. For example

```
search "Bookshop" 'o'  ==  [1,2,6]
search "senselessness" 's'  ==  [0,3,7,8,11,12]
```

Your definition should use a *list comprehension*. You may use the function `zip :: [a] -> [b] -> [(a,b)]`, the function `length :: [a] -> Int`, and the term forms `[m..n]` and `[m..]`.

(b) Try to come up with a property of `search` that should always hold. Write a QuickCheck test to confirm it does.

3 Really optional and unassessed, just for fun

Exercise 7

The puzzle game HaskellQuest, produced by Maxim Despinoy in 2022/2023 as his fourth-year project, provides a fun way of learning about list comprehensions in Haskell. You can access the game through the link https://uoe-my.sharepoint.com/:f:/g/personal/dts_ed_ac_uk/EqRa36_-wYlOnEkOwbQJTX0Bv0aGm9vk5un06vafzqneHA?e=USaLuS. See if you can solve the mystery! Unfortunately, the game only works in Windows.