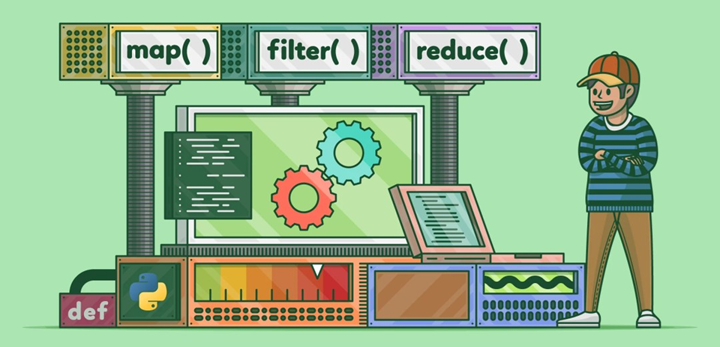
Functional Programming

5CM524

****

Lab Instructions for 1 hour lab  
of Week 2

Dr Youbing Zhao

Prof Stephan Reiff-Marganiec

# Aims

The attached instructions form content for 2 hours of labs. This lab contains no assessed element.

You’ll be trying your hand at some larger programmes this time:

1. Working with strings
2. Caesar Cypher encoding and decoding

# Overview

In this lab you will be writing some more Haskell functions and slightly larger programmes.

Just a reminder, we are using an Azure lab with the Haskell compiler preinstalled. The Azure lab has already been setup for you and there are instructions in the lab folder for last time of how to connect. Please make sure that you stop the VM when you are done as otherwise time will keep ticking down!

Note: If you work on your own machine then you can remote connect to the Azure lab or install Haskell from <https://www.haskell.org/ghc/download.html>. Install takes a little while!

# Let’s write a simple-ish string manipulation programme

Haskell has quite a few functions build in, but it is a good idea to write some as it helps you understand how to write functions and work with some of the basic structures.

A Palindrome is a word or sentence that reads the same forward or backward (ignoring spaces, capitals and punctuation). For example, ‘madam’ or ‘A man, a plan, a canal: Panama!’

Task: Write a palindrome checker

Goal: A programme that can check whether a string is a palindrome.

Hints:

I like you to approach this in the following way: first off assume that no punctuation is entered in the input string and also that everything is lower case; strip spaces out and then reverse the string. Finally, compare the reversed string with the original one. So, the functions you need are:

**reverse :: String -> String**

**stripSpaces :: String -> String**

**isPalindrome :: String -> Bool**

Note that we have seen **reverse** in the lecture; also remember that a string is the same as [Char] (a list of characters which has the x:xs structure) and is part of the **Eq** class. **:** is the list constructor that attaches an element at the front of a list; ++ is a list append operator that attaches a list at the end of a list.

# Caesar Cypher

We started to look at this in the lecture. Just as a reminder: a Caesar Cypher is a very simple encryption technique for text. It has been used in Roman times already for safe communications. Essentially it shifts every letter in the clear text by a number of steps to find the letter in the cyphertext; the alphabet warps around at the end/ start for this. So, shifting by 2 would move a ‘b’ to a ‘d’ and ‘z’ to a ‘b’. Shifts could be positive or negative moving right or left along the alphabet. Typically only letters get encrypted. Decryption just needs to shift text back by the same distance.

Task: Write a Ceaser encryption and decryption function.

Goal: A programme that can take a string and encrypt or decrypt it, depending on which function is called.

Hints:

You should import some functions from the Haskell libraries. So put the following at the start of the programme source code:

**import Data.Char (chr, ord, isAlpha, isUpper, isLower)**

**chr** converts an ASCII code to a character (e.g. **chr 97 = ‘a’**), **ord** gives the ASCII code for a character; **isAlpha** checks if the input character is an alphabetic character, **isUpper** and **isLower** should be self-explanatory.

We discussed **map** in lecture 3: **map** takes a function and a list and returns a list; it will apply the supplied function to all elements in the list.

I suggest a strategy of having a function that shifts a character by n steps, functions to encode and decode a single character using the shift function and then a function that applies the character encoding to a string. (You are of course welcome to follow another approach.)