Functional Programming	WS 2022	LVA 703025
Exercise Sheet 8, 10 points		Deadline: Wednesday, November 30, 2022, 6am

- Mark your completed exercises in the OLAT course of the PS.
- Start from `template_08.hs` provided on the proseminar page.
- Your .hs-file should be compilable with `ghci` and be uploaded in OLAT.

Exercise 1 *Le Chiffre Indéchiffrable* **5 p.**

The Vigenère cipher remained unbreakable for over 300 years, earning it the name "the indecipherable cipher":[1] your task in this exercise is to complete a Haskell program to break it. Messages can be encrypted and decrypted using a keyword and a Vigenère square (Figure 1). The template file already contains a function `vigenere` to encrypt and decrypt Vigènere messages with a known key.

A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 1: The Vigenère square. The keyword is repeated for the length of the message, and each letter of the message is encoded using the row of the message character and the column of the key character. For example, for message `HELLOWORLD` and keyword `GHCUP`, the first letter of the encrypted message is the letter at row `H` and column `G`, which is `N`. The second encrypted letter is `L` (at row `E` and column `H`), and the entire encrypted message is `NLNFDCVTFS`. To decrypt the message, this process is reversed: column `G` has letter `N` at row `H`, so the first decrypted letter is `H`.

1. (a) Using the `Prelude` function `map`, write a function `freqs :: String -> [Int]` that calculates the frequencies of uppercase letters in a string. Other characters should be ignored. (1 point)

   Examples:
   ```
   freqs "AABCCCZ" = [2,1,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
   freqs "Alice??" = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
   ```

[1] Friedrich Kasiski first published a method of breaking the cipher in 1863, but Charles Babbage independently broke the cipher as early as 1854. See *The Code Book* by Simon Singh for an exciting introduction to classical cryptography.

Hint: A string of all 26 uppercase English letters can be generated with `['A'..'Z']`.

(b) The index of coincidence (IoC) measures how likely it is that two random letters in a string are identical. For a string of length n with letter frequencies $f_0, \ldots, f_{25}$ for characters A, ..., Z, the IoC is:

$$\frac{1}{n(n-1)} \sum_{i=0}^{25} f_i(f_i - 1)$$

Write a function `indexOfCoincidence` to calculate the IoC of a string by using `foldr` on the letter frequencies. You may assume that the input string contains only characters in `['A'..'Z']`. (1 point)

Examples:
```
indexOfCoincidence "AAB" = 0.33333334
indexOfCoincidence "" -- *** Exception: ... -- length < 2
```

(c) Implement a function `nSubstrings :: String -> Int -> [String]` which returns the list of the n-substrings of a string, created by taking every n-th character of the string starting from index 0, $\ldots, n-1$. (1 point)

Examples:
```
nSubstrings "HASKELLISFUN" 0 = []
nSubstrings "HASKELLISFUN" 1 = ["HASKELLISFUN"]
nSubstrings "HASKELLISFUN" 5 = ["HLU","ALN","SI","KS","EF"]
```

Hint: `zip "HASKELLISFUN" [0..11]` returns a list `[("H", 0), ("A", 1), ("S", 2), ...]` which can then be used, for example, in a list comprehension to filter characters by index.

(d) Write a function `rotate` to wrap a list by `n` places to the left when $n > 0$. The type of `rotate` should be as general as possible. If $n \leq 0$ or n is greater than the length of the list, then the original list should be returned. (1 point)

Examples:
```
rotate 0 "Babbage" = "Babbage"
rotate 1 "Babbage" = "abbageB"
rotate (-2) "Babbage" = "Babbage"
```

Hint: Some of the library functions mentioned on the slides from lecture 8 might be helpful.

2. The function `keywordLen` finds a likely keyword length using the index of coincidence.[2] There are now 26 possibilities for each keyword letter, which we can guess using observation that some letters appear more frequently than others in English text.

The function `chisqr :: [Float] -> [Float] -> Float`, given a list of observed frequencies *os* and expected frequencies *es*, computes the *chi-square statistic*:

$$\sum_{i=0}^{n-1} \frac{(os_i - es_i)^2}{es_i}$$

Smaller results of the chi-square statistic indicate better matches between observed and expected frequencies. Write a function `chisqrVals :: String -> [Float]` which first computes the frequency percentages of letters in the string, then calculates the chi-square statistic of each possible rotation of the frequency list with respect to the average percentage frequencies in English text given in `englishFreqs`. (1 point)

Examples:
```
chisqrVals "THIS IS ENGLISH" = [164.09297, 346.37234, 6324.823,...
chisqrVals "UIJT JT FOHMJTI" = [3633.3218, 164.09297, 346.37234,...
```

The function `crackVigenere` uses the most likely keyword letter values from `chisqrVals` to guess the keyword. Try running `putStrLn $ crackVigenere ciphertext`. What happens?

Exercise 2 *Pages of text, type class instances, one-time pads, redaction* **5 p.**

Consider the datatype `Page`, representing pages of text by lists of strings, together with a `Show`-instance and a test page:

[2] William F. Friedman discovered that substrings taken every n characters tend to have a higher index of coincidence when n divides the keyword length: the function `keywordLen` finds the value of n in 1, ..., 8 with the highest average index of coincidence.

```haskell
data Page = Page [String]

instance Show Page where show (Page p) = unlines p

testPage = Page ["Dear Page,","","Please show us","some lines!","","Sincerely,","X."]
```

Moreover, consider the type class `ZPM` that encompasses types that have a zero value and support the operations plus (`<+>`) and minus (`<->`).

```haskell
class ZPM a where
  zero :: a
  (<+>), (<->) :: a -> a -> a
```

Instances of `ZPM` should typically satisfy the following three equations for arbitrary values of x:

$$x \texttt{ <+> } \texttt{zero} = x \qquad \texttt{zero} \texttt{ <+> } x = x \qquad x \texttt{ <+> } x \texttt{ <-> } x = x \qquad (\star)$$

1. Implement a function `pageOf :: String -> Page` that turns a string into a page by splitting it into lines.

   Example: `pageOf "This is\na test." = Page ["This is", "a test."]` (1 point)

2. Give an instance of type class `ZPM` for `Char`, such that the equations ($\star$) from above are satisfied. (1 point)

   Hint: You may find the type class `Enum` and/or `Data.Bits.xor` (for a bitwise XOR operation on integers) useful.

3. Assuming `ZPM a`, give a `ZPM`-instance for lists of type `[a]`, such that the length of the result of `xs <+> ys` is the longer of the lengths of `xs` and `ys` and the equations ($\star$) from above are satisfied. (1 point)

   Example: `length ("test" <+> "x") = 4`

4. Give a `ZPM`-instance for `Page` satisfying the equations ($\star$) from above. (1 point)

   Motivation: This `ZPM`-instance allows us to apply one-time pads[3]—represented by a page `k` (for *key*)—to a page of plain text `p` by performing the operation `p <+> k`. The resulting encrypted page `q` can be decrypted by performing `q <-> k`. Try it, by applying a key of your choosing to the test page from above.

5. Implement a function `redact :: String -> Page -> Page` such that `redact w p` redacts all occurrences of the word `w` on the page `p` by replacing each character of `w` by `'X'`. (1 point)

   Examples: `redact "Page" testPage =`
   ```haskell
     pageOf "Dear XXXX,\n\nPlease show us\nsome lines!\n\nSincerely,\nX."
   redact "haha" (pageOf "hahaha") = pageOf "XXXXha"
   redact "haha" (pageOf "hahahaha") = pageOf "XXXXXXXX""
   ```

[3] https://en.wikipedia.org/wiki/One-time_pad