Development 8 - Exercises
Unit 2

For these exercises use the data structure `List<T>` defined in the library `immutable.js`. To add the library to your typescript project type

```
yarn add -D immutable
```

in the shell. Then add on top of your source code

```
import * as Immutable from "immutable"
```

Do not use functions from that library that immediately solve the exercises. Try to implement everything from scratch and just use lists as a data structure.

Exercise 1:

Implement a function

```
let last = <a>(l:  Immutable.List<a>):  a
```

that returns the last element of a list.

Exercise 2:

Implement a function

```
let rev = <a>(l:  Immutable.List<a>):  Immutable.List<a>
```

that creates a list with the elements of `l` in reverse order.

Exercise 3:

Implement a function

```
let append = <a>(l1:  Immutable.List<a>) => (l2:  Immutable.List<a>):
Immutable.List<a>
```

that adds all the elements of `l2` after those in `l1`.

Exercise 4:

Implement a function

```
let nth = <a>(n:  number) => (l:  Immutable.List<a>):  a
```

that returns the element in position `n` in `l`.

Exercise 5:

Implement a function

```
let palindrome = <a>(l:  Immutable.List<a>):  boolean
```

that checks if a list is palindrome. A list is palindrome if it is equal to its inverse.

Exercise 6:

Implement a function

```
let palindrome = <a>(l:  Immutable.List<a>):  boolean
```

that checks if a list is palindrome. A list is palindrome if it is equal to its inverse.

Exercise 7:

Implement a function

```
let compress = <a>(l:  Immutable.List<a>):  Immutable.List<a>
```

that removes consecutive occurrences of the same element in the list. For example `compress [a;a;a;a;b;b;c;c;b] = [a;b;c;b]`.

Exercise 8:

Implement a function

```
let caesarCypher = (l:  Immutable.List<string>) => (shift:  number):
Immutable.List<string>
```

The Caesar's cypher take a text, represented as a list of characters (note that Typescript does not support the type `char` so you can use a list of `string` with only one character), and shifts all the letters (so only if the character is an alphabetical character) in it up by the number of position specified by `shift`. If the letter goes past `z` it restarts from `a`. You can assume that all the text is in lower-case letter. For instance:

```
shift("c")(5) = h
shift("y")(5) = d
```

The ASCII code for a specific character in a `string` can be obtained by using the method `charCodeAt` that takes as input the position of the character of the string you want to get the ASCII code of. For instance:

```
"Caesar".charCodeAt(2) = 101
```

Advanced: Try to support also upper-case letters in the text.