# Haskell & Functional Programming

## Exercise 1

Write a function `myReplicate` that given an integer `n` and a value `v` returns a list of *length* `n` initialized with `v`, namely all elements are equal to `v`.

- **Goal:** Warming up!
- **Expected output:** Two implementations of `myReplicate`: one recursive and one using the combinators `map`, `filter`, `foldl/r` from the Haskell Prelude.

## Exercise 2

Write a function `sumOdd` that given a list of integers computes the sum of the values that are *odd*.

**Hint**: consider the functions `odd` and `even` of the `Prelude`.

- **Goal:** Warming up (pt. 2)!
- **Expected output:** Two implementations of `sumOdd`: one recursive and one using the combinators `map`, `filter`, `foldl/r` from the Haskell Prelude.

## Exercise 3

Write a function `repl` that given a list `xs` and a integer `n` returns a list containing the elements of `xs` replicated `n` times.

**Hint**: you can use the function `createList` of Exercise 1.

- **Goal:** Playing with lists.
- **Expected output:** Two implementations of `repl`: one recursive and one using the combinators `map`, `filter`, `foldl/r` from the Haskell Prelude.

## Exercise 4

Write a function `totalLength` that given a list of strings `xs` computes the sum of the lengths of the strings starting with the character '`A`'.

- **Goal:** Test your skills with lists and strings.
- **Expected output:** Two implementations of `totalLength`: one recursive and one using the combinators `map`, `filter`, `foldl/r` from the Haskell Prelude.

## Exercise 5

Write a function `filterOdd` that given a list `xs` returns a new list obtained from `xs` by removing the elements at odd positions.

**Hint**: Here "odd positions" means the first, third, fifth, etc position.

- **Goal:** Playing with lists (pt. 2).
- **Expected output:** Two implementations of `filterOdd`: one recursive and one using the combinators `map`, `filter`, `foldl/r` from the Haskell Prelude.

# Exercise 6

Write a function `titlecase` that given a string `s` converts it to *titlecase* by uppercasing the first letter of every word.

**Hint**: consider using the function `words`, `unwords` of the `Prelude` and the function `toUpper` of the module `Data.Char`. To make accessible this last function in your code use `import Data.Char (toUpper)`.

- **Goal:** Experimenting with strings.
- **Expected output:** Two implementations of `titlecase`: one recursive and one using the combinators `map`, `filter`, `foldl/r` from the Haskell Prelude.

# Exercise 7

Write a function `countVowelPali` that given a list of strings `xs` returns the total number of vowels in strings that are palindromes. For example,

```
countVowelPali ["anna", "banana", "civic", "mouse"] = 4
```

- **Goal:** Fun with strings and lists (again :P).
- **Expected output:** Two implementations of `countVowelPali`: one recursive and one using the combinators `map`, `filter`, `foldl/r` from the Haskell Prelude.

# Exercise 8

Recall the higher-order combinator `map` from the `Prelude`. Implement it using the combinator `foldl`.

- **Goal:** Experimenting with combinators.
- **Expected output:** A file containing the required implementation of the `map` combinator.

# Exercise 9

Consider the following definition of binary trees:

```
data IntTree = Leaf Int | Node (Int, IntTree, IntTree)
```

1. Implement `tmap`, a "tree version" of the `map` combinator. More precisely, the function `tmap` should take a function `f` and a tree `t` and should apply `f` to each value in `t`.
2. Using `tmap` implement the function `succTree` taking a tree `t` and computing a tree whose elements are the successors of the values in `t`.
3. Write a function `sumSucc` taking a tree `t` and computing the sum of the elements of `succTree t`.

- **Goal:** Experimenting with trees.
- **Expected output:** A file containing the three required functions.

# Exercise 10

Implement a tail recursive version of the `map` and `filter` combinators.

- **Goal:** Trying to write tail recursive functions.
- **Expected output:** A file containing the required combinators.

# Exercise 11

Read the web page Foldr Foldl Foldl'. Write some minimal examples highlighting the differences between the three functions.

- **Goal:** Exploring alternative implementations of popular combinators.
- **Expected output:** A file containing the required examples.

Author: Andrea Corradini & Matteo Busi

Created: 2018-11-12 lun 06:27

Validate