Functional Programming for BDA - List 1
Recursion and lists

Marcin Michalski, DPM FPAM WUST 2021/2022

17.10.2021

If not said otherwise the exercises should be done via recursion and basic operations on lists, including the list comprehension and excluding `fmap` and `fold` of any sort. The latter will be a topic of a future list.

Exercise 1. Upgrade the following implementation of the factorial so that it is tail recursive and *fast*

```
factorial 0 = 1
factorial n = n * (factorial (n-1))
```

Exercise 2. Upgrade the following implementation of list reversing so that it is tail recursive and *fast*

```
rev [] = []
rev (x:xs) = (rev xs) ++ [x]
```

Exercise 3. Implement a function that for a given natural n quickly counts the amount of zeros at the end of $n!$

Exercise 4. Implement your own functions that curry and de-curry functions, i.e. for $f \in C^{(A\times B)}$ and $g \in (C^B)^A$

`(my_curry f) a b = f (a,b)`,
`(my_decurry g) (a,b) = g a b`.

Exercise 5. Implement the sieve of Eratosthenes. Your solution should be fast.

Exercise 6. The Euler's totient function $\varphi : \mathbb{N}_+ \to \mathbb{N}$ is defined as follows

$$\varphi(n) = |\{k \in \mathbb{N}_+ : k \leqslant n \ \& \ \gcd(k, n) = 1\}|.$$

Implement

(a) the Euler's totient function.

(b) a function $f(n) = \sum_{d\in\{k\in\mathbb{N}_+ : k|n\}} \varphi(d)$. (*) Put forward a hypothesis and try to prove it.

Exercise 7. (a) Implement a function that calculates the n-th member of Fibonacci sequence in a linear time.

(b) The same for the sequence

$$a_0 = 1,$$
$$a_1 = 1,$$
$$a_n = n + a_{n-1} + a_{n-2}.$$

Exercise 8. See the documentation of the function `zipWith`. Create an infinite list list of Fibonacci numbers using zipWith and lazy evaluation.

Exercise 9. Implement a function

(a) `ecd` that for a given string (a list of chars) eliminates consecutive duplicates, i.e.

```
ecd [1,1,2,3,3] == [1,2,3].
```

(b) `encode` that for a given string encodes consecutive duplicates with an integer, i.e.

```
encode [a,a,a,b,b,a,a] == [(a,3), (b,2), (a,2)].
```

(c) `decode` that decodes the previous one, i.e.

```
decode (encode xs) == xs.
```

Exercise 10. Implement a function `rev_rev` that for a list of string returns the reversed list of reversed strings, i.e.

```
rev_rev ["abc","xyz"] == ["zyx","cba"].
```

Exercise 11. Implement a function `substrings` that for a given string returns the list of all its substrings, i.e.

```
substrings "abc" == ["a","b","c","ab","bc","abc"].
```

Exercise 12. Implement a function `power_list` that for a given list returns the list of all its sublists, i.e.

```
power_list [1,2,3] == [[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]].
```

Exercise 13. Implement a function `perm` that for a given list returns the list of all its permutations, i.e.

```
perm [1,2,3] == [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]].
```

You may assume that the list is without duplicates.