(OPTIONAL) Functional Programming - Bonus Exercise

This exercise sheet is optional!

- **Exercise 1: Aggregation deep dive** We looked at simple implementations of the fold function. You might have noticed, that there is also a foldBack function.
    - Implement a variant of List.map. Instead of writing the recursion yourself, utilize `fold` or `foldBack` and provide it with higher-order-functions which take care of building the result list. If your solution does not feel switch the fold variant. HINT: the state will be of type `list` , and the fold fold function `('s -> 'a -> 's)` needs to add an element to the state. Attention: `foldBack` and `fold` has a slightly different signature (the parameters are switched in the folding function).
    - In the snippet below are the definitions of foldl and foldr (the haskell names for fold and foldBack). As we have seen in the previous task, they behave differently when it comes to the order of putting together intermediate aggregate values. Research the term `tail call optimization` and argue which will typically run faster.

```
// foldl f z [x1, x2, ..., xn] == (...((z `f` x1) `f` x2) `f`...) `f` xn
// List.fold in F#
let rec foldl (acc : 's) (f : 'a -> 's -> 's) (xs : list<'a>) : 's =
    match xs with
    | [] -> acc
    | x::xs -> foldl (f x acc) f xs

// called foldr in haskell
// foldr f z [x1, x2, ..., xn] == x1 `f` (x2 `f` ... (xn `f` z)...)
// List.foldBack in F#
let rec foldr f s xs =
    match xs with
    | [] -> s
    | x::xs -> f x (foldr f s xs)
```

- **Exercise 2: Immutable Map - Adding values** Given the file `ImmutableMap.fsx` , implement the function to add an element to the (unbalanced) binary search tree. Seems easy right, but useless since it does no balancing. Take a look at a red-black-tree implementation here.

If you have additional questions, add it to the submission and i will try to answer them.

- **Sumbission.** Submit your as condensed as possible - e.g. a single file with all the code/markdown. Please don't put it into a zip folder if possible.