Functional Programming

http://proglang.informatik.uni-freiburg.de/teaching/functional-programming/2017/

Exercise Sheet 4 – High order functions, IO

2017-11-29

1 High order functions

Exercise 1 (Folding)
Fold is a very common functional programming idiom:

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

1. Define `foldr`.
2. Using `foldr`, implement:
   - `or`, returns `True` if at least one item in the list of booleans is `true`
   - `filter`
   - `map`
   - `foldl`, the left-associative variant of `foldr`:

     ```
     foldl :: (b -> a -> b) -> b -> [a] -> b
     foldl _ acc [] = acc
     foldl f acc (x: xs) = foldl f (f acc xs) xs
     ```

   - `remdups`, removes consecutive duplicates from a list

Exercise 2 (Improving the Vector Graphics Library)
In your vector graphics library, implemented during exercise sheet 3, you might have distinguished the type of coordinates:

```
data Coord = Coord Float Float
```

Which is the same as the datatype for 2D Vectors.

1. Write the following high order functions that manipulates pictures:
   - `mapCoord :: (Coord -> Coord) -> Picture -> Picture`.
     `mappCoord f pic` applies `f` to all the coordinates inside `pic`.
   - `mapScalar :: (Float -> Float) -> Picture -> Picture`.
     `mapScalar f pic` applies `f` to all the scalar values inside `pic` (such as radius of circles).
2. Use these functions to implement `move`, `scale` and `rotate`. These implementations should now be very short (only one line).

Exercise 3 (Unfolding)
There is also a dual function to `foldr`, `unfoldr`:

```
unfoldr :: (b -> Maybe (a, b)) -> b -> [a]
```

Instead of reducing a list to a final result, `unfoldr f seed` builds a new list: The elements of the list are created by repeatedly applying the `f` function to the accumulator `b`. If `f b` returns the value `Nothing`, the list is over. If `f b` returns the value `Just (a, b')`, then `a` is added as the foremost element. The value `b'` is then passed to `f` to calculate the next element.

1. Define `unfoldr`.
2. Using `unfoldr`, define `map`.
3. Another standard function of functional programming is `iterate :: (a -> a) -> a -> [a]`
   What could this function do? Implement `iterate` using `unfoldr`.

2 The IO type

Exercise 4 (Numbers game)
In the Numbers game, the computer tries to guess a user-imagined number between 1 and 100. Here is an example. Texts after the `>` prompt are user inputs.

```
Choose a number between 1 and 100!
Is it 50?
> greater
Is it 75?
> smaller
Is it 62?
> smaller
Is it 56?
> Yes
I won in 4 attempts!
```

Implement this game using the `IO` type and the `do` notation. In your Stack project, add this program in the `/bin` directory to easily produce a real binary.

Exercise 5 (Stack Calculator Interface)
In exercise sheet 1, we implemented a simple stack calculator. This calculator was missing a crucial component: a command line interface!

Using `IO`, add a command line interface to your implementation of the stack calculator. Each line should represent a command (for example "push 3" or "add"). The program should show the stack at each step. "exit" should exit the program.