Functional Programming for BDA - List 4

Maybe and non-determinism, `do` notation

Marcin Michalski, DCS FFPT WUST 2019/2020

Treat Exercises as a warm up. Submit Tasks 1, 2 and 3.

Exercise 1. Let `f x = [x+1,x+2]` and `g x = [2*x,3*x]`. Understand and calculate `[1,1,1] >>= f` and `([1,1,1] >>= f) >>= g`.

Exercise 2. Simplify `do x <- mx; f x`. What should be the type of `f`?

Exercise 3. Explain how the `do` notation makes the list comprehension redundant.

Task 1. Implement a model of ”walking a narrow path”. A ”wanderer” starts at a position `pos` (an integer satisfying $-3 <$ `pos` < 3) and moves forward and left or forward or forward and right (which changes the wanderer's position by -1, 0, 1 respectively). If the wanderer wanders too much to one of the sides of the path, he dies (`|pos|` > 2). Implement

a) a function `move :: Int -> Int -> Maybe Int` that takes a move and a position and returns the new position (if the wanderer lives) or `Nothing` (if he dies). Use `>>=` to make a couple of moves,

b) a function `move_list :: [Int] -> Int -> Maybe Int` that does almost the same thing, however it takes a list of moves instead of one move,

Task 2. Implement a function that returns a list of all the possible outcomes of two (d6 and d20) dices roll. Use `do` notation or `>>=`.

Task 3. Implement a function that takes a starting position of a knight on a chess board of size $n \times k$ and returns a list of its possible positions in

a) 3 moves,

b) any number of moves.

Use `>>=` or `do` notation.