Functional Programming for BDA - List 4
Maybe and non-determinism, `>>=` and `do` notation

Marcin Michalski WUST 2021/2022

09.11.2022

Exercises are for you to better understand concepts on this list. Submit Tasks 1 and 2 only.

Exercise 1. Let `f x = [x+1,x+2]` and `g x = [2*x,3*x]`. Examine and calculate `[1,4,7] >>= f` and `([1,4,7] >>= f) >>= g`.

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

Exercise 3. Desugar `do x <- mx; f x`. What should be the type of `f`?

Exercise 4. Show that the following pieces of code

(i) `do f <- mf; x <- mx; return (f x);`

(ii) `do f <- mf; fmap f mx;`

are equivalent.

Exercise 5. Explain how the `do` notation makes the list comprehension redundant.

Exercise 6. Can `join` be defined with `return` and `>>=` operator?
Consider and simplify the following piece of code

```
fun mmx = do
  mx <- mmx
  x <- mx
  return x
```

Exercise 7. Identify `>>=`, `>=>`, and `>>` for monads you are familiar with (you *should* be familiar with at least two :-)).

Exercise 8. Implement a model of "walking a narrow path". The *wanderer* starts at a position `pos` (an integer satisfying $-3 <$ `pos` < 3) and moves forward and left or forward and right with each move (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 $\in \{-1, 0, 1\}$ and a position and returns the new position (if the wanderer lives) or `Nothing` (if he dies). Use `>>=` to make a couple of moves. Examples of outcomes:

```
move 1 (-1) = Just 0,  move 1 2 = Nothing
```

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, e.g.

```
move_list [1,1,0,-1] 1 = Nothing,  move_list [1,0,-1,-1] 1 = Just 0.
```

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

Exercise 9. 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, i.e. the number of moves is the function's argument.

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