Functional Languages 9th practice

1. Redefine function `map`.

```
map' (\n -> n + 2) [] == []
map' (\n -> n + 2) [2,3,4] == [4,5,6]
map' even [2,3,4] == [True, False, True]
```

2. Redefine function `filter`.

```
filter' (\n -> n > 5) [] == []
filter' (\n -> n > 5) [1,2,5,6,0] == [6]
filter' even [1,2,5,6,0] == [2,6,0]
filter' (elem 0) [[5,6],[4,1,2,0],[0,5]] ==
  [[4,1,2,0],[0,5]]
```

3. Define a function `upperToLower` which converts every capital letter to lower case and removes the rest.

```
upperToLower "" == ""
upperToLower "Hello World!" == "hw"
upperToLower "haSKell" == "sk"
```

4. Redefine function `all`.

```
all' (\n -> n > 0) []
all' (\n -> n > 0) [1,2,9,6]
not (all' (\n -> n > 0) [1,2,-4,9,6])
not (all' even [4,6,8,2,3,0])
```

5. Redefine function `any`.

```
not (any' (\n -> n == 2) [])
not (any' (\n -> n == 2) [1,5,9,0,3])
any' (\n -> n == 2) [1,5,9,2,0,3,2]
any' even [1,2,5,9,2,0,3,2]
```

6. Define a function `hasLongLine` which checks one of the lines of a file has at least 3 words.

```
not (hasLongLine "first\nsecond\nthird line")
hasLongLine "first\none fat line\nthird line"
```

   Hint: here `lines` and `words` offer a help.

7. Redefine function `elem`, which checks whether an element is in a list. Use higher order functions.

```
elem' :: Eq a => a -> [a] -> Bool

elem' 'n' "Finn"
elem' 'H' "Harry"
not (elem' 'h' "Harry")
elem' True [False, False, True]
not (elem' True [False, False, False])
```

8. Define a function `hasAny` that checks whether an element in a list occurs in another list too.

```
hasAny :: Eq a => [a] -> [a] -> Bool

hasAny "abc" "I like Haskell"
hasAny [5,9] [4, 3, 2, 0, 9]
not (hasAny ["haskell", "python"] ["c", "java"])
```

9. Redefine function `takeWhile`.

```
takeWhile' (\n -> n > 5) [] == []
takeWhile' (\n -> n > 5) [6,7,9,5,2,1] == [6,7,9]
takeWhile' odd [6,7,9,5,2,1] == []
takeWhile' odd [7,9,5,2,1] == [7,9,5]
```

10. Redefine function `dropWhile`.

```
dropWhile' (\n -> n > 5) [] == []
dropWhile' (\n -> n > 5) [6,7,9,5,2,1] == [5,2,1]
dropWhile' odd [6,7,9,5,2,1] == [6,7,9,5,2,1]
dropWhile' odd [7,9,5,2,1] == [2,1]
```

11. Define function `dropWord` which removes the first word from the beginning of a string.

```
dropWord "" == ""
dropWord " tree " == " tree "
dropWord "apple tree " == " tree "
dropWord "appletree" == ""
```

12. Given a username and a database of usernames and passwords, check whether the database contains the username.

```
users :: [(String, String)]
users = [ ("mrbean", "4321")
        , ("admin", "s3cr3t")
        , ("finn", "algebraic")
        ]

doesUserExist "admin" users
doesUserExist "finn" users
not (doesUserExist "darth_vader" users)
```