19CSE313 Principles of Programming Languages Practice Questions

R.Abhinav

CB.EN.U4CSE19453

**Write the functions given below. Check whether your function works for both list of numbers and strings. You can try writing the functions below both in *Haskell* and *Scala* to compare and appreciate the functional implementation. This will give you a perspective on functional programming from different languages.**

1. Write a function rember that accepts a list and an element of the list and returns the list with the first occurence of the element in the list removed. For example rember 10 [1,**10**,100,1000] =» [1,100,1000] and rember “l” “Haske**l**l” =» “Haskel”. Can you write remberAll that removes all occurrences of the elements? For example, remberall 10

[1,**10**,100,**10**,1000] =» [1,100,1000].

**Code:**

rember :: Eq t => t -> [t] -> [t]

rember y [] = []

rember y (x : xs)

  | x == y = xs

  | otherwise = x : rember y xs

*--b*

remberall :: Eq a => a -> [a] -> [a]

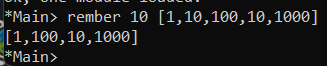
remberall y [] = []

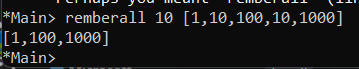
remberall y (x : xs)

  | x == y = remberall y xs

  | otherwise = x : remberall y xs

**Output:**

****

****

1. Write a function firsts that accepts a list of lists and returns a list of all the first elements of the sublists. For example firsts [[**1**,10],[**10**,100], [**100**,1000] =» [1,10,100]. Similarly can you write the seconds function? Seconds behaves as follows. Seconds [[1,**10**],[10,**100**],[100,**1000**] =» [10,100,1000].

**Code:**

firsts :: [[a]] -> [a]

firsts [] = []

firsts (x : xs) = head x : firsts xs

main = do

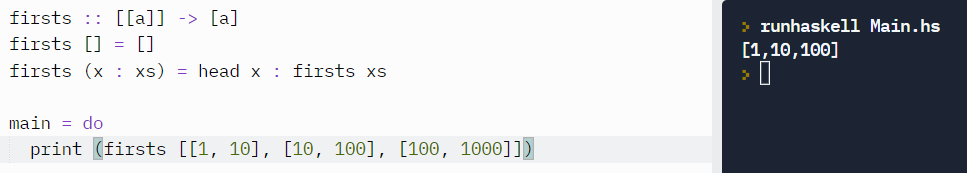
  print (firsts [[1, 10], [10, 100], [100, 1000]])

*--b*

seconds :: [[a]] -> [a]

seconds x = [head (tail xs) | xs <- x]

**Output:**

****

1. Write a function insertRight that accepts three arguments viz. an element in the list, a new element to be inserted to the right side of the first occurrence of the given element and a list. For example insertRight 100 1000

[1,10,100,10000,100000] =» [1,10,100,**1000**,10000,100000]. Also,

write the insertLeft function whose meaning is obvious. InsertLeft 100

10 [1,100,1000] =» [1,**10**,100,1000]. Can you write the

insertRightAll and insertLeftAll which inserts for all occurrences of the element in the list. InsertRightAll 100 1000 [1,10,**100**,10000,**100**] =»

[1,10,100,**1000**,10000,100,**1000**].

**Code:**

*--insertRight*

insertRight :: Eq t => t -> t -> [t] -> [t]

insertRight y a [] = []

insertRight y a (x : xs)

  | x == y = x : a : xs

  | otherwise = x : insertRight y a xs

*--insertRightall*

insertRightall :: Eq a => a -> a -> [a] -> [a]

insertRightall y a [] = []

insertRightall y a (x : xs)

  | x == y = x : a : insertRightall y a xs

  | otherwise = x : insertRightall y a xs

*--insertLeft*

insertLeft :: Eq t => t -> t -> [t] -> [t]

insertLeft y a [] = []

insertLeft y a (x : xs)

  | x == y = a : x : xs

  | otherwise = x : insertLeft y a xs

*--insertLeftall*

insertLeftall :: Eq a => a -> a -> [a] -> [a]

insertLeftall y a [] = []

insertLeftall y a (x : xs)

  | x == y = a : x : insertLeftall y a xs

  | otherwise = x : insertLeftall y a xs

**Output:**

1. Write a function replace that accepts an element in the list, a new element to replace the first occurence of the given element and list. For example replace 10 100 [1,10,100,1000] =» [1,**100**,100,1000]. Can you write a replaceAll function which replaces all occurrences of the given element?

**Code:**

replace :: Eq t => t -> t -> [t] -> [t]

replace y a [] = []

replace y a (x : xs)

  | x == y = a : xs

  | otherwise = x : replace y a xs

*--b*

replaceAll :: Eq a => a -> a -> [a] -> [a]

replaceAll y a [] = []

replaceAll y a (x : xs)

  | x == y = a : replaceAll y a xs

  | otherwise = x : replaceAll y a xs

**Output:**

1. Write a function replaceEitherOr that accepts two elements in the list, a new element to replace the first occurrence of either of the two lists whichever comes first and a list. For example replaceEitherOr 10 100 1 [1,**10**,100,1000]

=» [1,**1**,100,1000]

**Code:**

replaceEitherOr :: Eq t => t -> t -> t -> [t] -> [t]

replaceEitherOr y a b [] = []

replaceEitherOr y a b (x : xs)

  | (x == y || x == a) = b : xs

  | otherwise = x : replaceEitherOr y a b xs

1. You know that elem function, given an element and a list, tells whether the element is present in the list or not. However you can check that elem function does not work in the case of a list of lists i.e. elem 5 [[1],[2],[3],[4],[5]] does not work. Can you write a version of elem that can check for the given element even if it is inside a sub-list of the given list.

**Code:**

element :: (Foldable t1, Eq t2) => t2 -> [t1 t2] -> Bool

element x [] = False

element x (y : ys)

  | elem x y = True

  | otherwise = (element x ys)