CptS 355 - Higher Order Functions – Class Exercises

1) `get_seconds`

Write a function "`get_seconds`" that takes a list of tuples and returns a list that includes the second elements of each tuple. Write a solution without using recursion explicitly.

Examples:

```
> get_seconds [(1,'H'),(2,'A'),(3,'S'),(4,'K'),(5,'E'),(6,'L'),(7,'L')]
"HASKELL"
```

2) `get_outof_range`

Define a function `get_outof_range` which takes two values, `v1` and `v2`, and a list "`xs`", and returns the values in `xs` which are less than `v1` and greater than `v2` (exclusive). **Your function shouldn't need a recursion but should use a higher order function** (`map`, `foldr/foldl`, or `filter`). You may need to define additional helper function(s), which are also not recursive.

Examples:

```
> get_outof_range (-5) 5 [10,5,0,1,2,-5,-10]
[10,-10]
> get_outof_range 4 6  [1,2,3,4,5,6,7,8,9,10]
[1,2,3,7,8,9,10]
> get_outof_range 'A' 'z' "CptS-355"
"-355"
```

Important note about negative integer arguments:

In Haskell, the -x, where x is a number, is a special form and it is a prefix (and unary) operator negating an integer value. When you pass a negative number as argument function, you may need to enclose the negative number in parenthesis to make sure that unary (-) is applied to the integer value before it is passed to the function.

For example: `get_outof_range -5 5 [-10,-5,0,5,10]` will give a type error, but
`get_outof_range (-5) 5 [-10,-5,0,5,10]` will work

3) `count_outof_range`

Define a function `count_outof_range` which takes two integer values, `v1` and `v2`, and a list "`xs`", and returns the total number of values in `xs` which are less than `v1` and greater than `v2` (exclusive). **Your function shouldn't need a recursion but should use higher order function** (`map`, `foldr/foldl`, or `filter`). You may need to define additional helper function(s), which are also not recursive.

Examples:

```
> count_outof_range (-5) 5 [10,5,0,1,2,-5,-10]
2
> count_outof_range 4 6  [1,2,3,4,5,6,7,8,9,10]
7
> count_outof_range 'A' 'z' "CptS-355"
4
```

4) nested_count_outof_range

Define a function `nested_count_outof_range` which takes two integer values, `v1` and `v2`, and a **nested** list "`xs`", and returns the total number of values in all elements of `xs` which are less than `v1` and greater than `v2` (exclusive). **Your function shouldn't need a recursion but should use higher order function** (`map`, `foldr/foldl`, or `filter`). You may need to define additional helper function(s), which are also not recursive.

Examples:

```
> nested_count_outof_range (-5) 5 [[10,5,0,1,2,-5,-10],[4,2,-1,3,-
4,8,5,9,4,10],[-5,-6,7,8]]
8
> nested_count_outof_range 'A' 'z' ["Cpt S","-","355",":","HW2"]
7
> nested_count_outof_range 1 1 [[4,1],[2,-1,3,-4],[8,0,1,5,9,4]]
10
```

5) find_routes

Assume the "`routes`" data given below.

```
routes = [
    ("Lentil", ["Chinook", "Orchard", "Valley", "Emerald","Providence", "Stadium",
      "Main", "Arbor", "Sunnyside", "Fountain", "Crestview", "Wheatland", "Walmart",
      "Bishop", "Derby", "Dilke"]),
    ("Wheat",["Chinook", "Orchard", "Valley", "Maple","Aspen", "TerreView", "Clay",
     "Dismores", "Martin", "Bishop", "Walmart", "PorchLight", "Campus"]),
    ("Silver",["TransferStation", "PorchLight", "Stadium", "Bishop","Walmart",
     "Outlet", "RockeyWay","Main"]),
    ("Blue",["TransferStation", "State", "Larry", "TerreView","Grand", "TacoBell",
     "Chinook", "Library"]),
    ("Gray",["TransferStation", "Wawawai", "Main", "Sunnyside","Crestview",
     "CityHall", "Stadium", "Colorado"]),
    ("Coffee",["TransferStation", "Grand", "Main", "Visitor","Stadium", "Spark",
     "CUB"])
   ]
```

Function `find_routes` takes the list of bus routes and a stop name, and returns the list of the bus routes which stop at the given bus stop.

Write the `find_routes` function **using higher order functions** (`map`, `foldr/foldl`, or `filter`) **and without using recursion.** Your helper functions should not be recursive as well, but they can use higher order functions. You can make use of `elem` function in your solution. The order of the elements in the output can be arbitrary.

Examples:

```
> find_routes "Walmart" routes
["Lentil","Wheat","Silver"]
> find_routes "Rosauers" routes
[]
> find_routes "Main" routes
["Lentil","Silver","Gray","Coffee"]
```