Problems

1. exists and countInList – 15%

a) **[5pts]** Write a function **`exists`** which takes a "value" and a "list" as input. If the value is a member of the list, the function should return `True`. Otherwise it should return `False`.
The function should have type `exists :: Eq t => t -> [t] -> Bool`.
(You are not allowed to use the `elem` function in your `exists` implementation.)

Examples:

```
> exists 1 []
False
> exists 1 [1,2,3]
True
> exists [1] [[1]]
True
> exists [1] [[3],[5]]
False
> exists '3' "CptS355"
True
```

b) **[2pts]** Explain in a comment why the type `is`

```
exists :: Eq t => t -> [t] -> Bool
```

but not

```
exists :: t -> [t] -> Bool
```

c) **[8pts]** Write a function **`countInList`** which takes a value and a list and returns the number of occurrences of that value in the input list.
The type of your function should be `countInList :: (Num p, Eq t) => t -> [t] -> p.`

Examples:

```
> countInList "5" ["3","5","5","-","4","5","1"]
3
> countInList "5" []
0
> countInList True [True, False, False, False, True, True, True]
4
> countInList [] [[],[1,2],[3,2],[5,6,7],[8],[]]
2
```

2. listDiff - 15%

Write a function `listDiff` that takes two lists as input and returns the difference of the first list with respect to the second. The input lists may have duplicate elements. If an element appears in both lists and if the number of duplicate copies of the element is bigger in the first list, then this element should appear in the result as many times as the difference of the number of occurrences in the input lists.

Your function should have type `listDiff :: Eq a => [a] -> [a] -> [a].` The duplicates should not be eliminated in the result. The elements in the resulting list may have arbitrary order.
(Hint: You can make use of `countInList` function in your solution.)

Examples:

```
> listDiff ["a","b","c"] ["b"]
["a","c"]
> listDiff [1,2,3] [1,1,2]
[3]
> listDiff [1,2,2,3,3,3] [1,1,2,3]
[2,3,3]
> listDiff [[2,3],[1,2],[2,3]] [[1],[2,3]]
[[2,3],[1,2]]
> listDiff [1,2,3] []
[1,2,3]
```

3. firstN – 15%

Write a function `firstN` that takes a list and a number `n` and returns the first `n` elements in the list. The type of `firstN` can be either of the following:

```
firstN :: (Num t) => [a] -> t -> [a]
firstN :: (Eq t, Num t) => [a] -> t -> [a]
firstN :: (Ord t, Num t) => [a] -> t -> [a]
```

If the input list is empty or if the length of the list is less than `n`, then the function should return the complete list. (You may assume that `n` is greater than 0.)

Examples:

```
> firstN ["a", "b", "c", "x", "y"] 3
["a", "b", "c"]
> firstN [1,2,3,4,5,6,7] 4
[1,2,3,4]
> firstN [1,2,3,4,5,6,7] 10
[1,2,3,4,5,6,7]
> firstN [[1,2,3],[4,5],[6],[],[7,8],[]] 4
[[1,2,3],[4,5],[6],[]]
> firstN [] 5
[]
```

4. busFinder – 20%

Pullman Transit offers many bus routes in Pullman. Assume that they maintain the bus stops for their routes as a list of tuples. The first element of each tuple is the bus route and the second element is the list of stops for that route (see below for an example).

```
buses = [("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", "Shopco",
          "RockeyWay"]),
         ("Blue",["TransferStation", "State", "Larry", "TerreView","Grand", "TacoBell",
          "Chinook", "Library"]),
         ("Gray",["TransferStation", "Wawawai", "Main", "Sunnyside","Crestview", "CityHall",
          "Stadium", "Colorado"])
        ]
```

Assume that you are creating an application for Pullman Transit. You would like to write an Haskell function `busFinder` that 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.
The type of `busFinder` should be `busFinder :: Eq t => t -> [(a, [t])] -> [a]`.
(Hint: You can make use of `exists` function you defined earlier.)

Examples:
```
> busFinder "Walmart" buses
["Lentil","Wheat","Silver"]

> busFinder "Shopco" buses
["Silver"]

> busFinder "Main" buses
["Lentil","Gray"]

> busFinder "Beasley" buses
[]
```

5. `cumulativeSums` – 15%
Write a function `cumulativeSums` that takes a list of numbers and returns a list including the partial sums of these numbers.
The type of `cumulativeSums` should be: `cumulativeSums :: Num a => [a] -> [a]`
(Hint: Define and use a helper function that takes a list and a number holding the accumulated sum.)
Examples:
```
> cumulativeSums [1,2,3,4,5,6,7,8,9,10]
[1,3,6,10,15,21,28,36,45,55]

> cumulativeSums [5,5,5,5,5,5,5]
[5,10,15,20,25,30,35]

> cumulativeSums [1,2,3,4,-4,-3,-2]
[1,3,6,10,6,3,1]

> cumulativeSums []
[]
```

6. `groupNleft` - 20%
`groupNleft` function takes two arguments where the first argument is a number (`n`) and the second is a list (`iL`). The goal is to produce a result in which the elements of the original list have been collected into ordered sub-lists each containing `n` elements (where `n` is the number argument). The leftover elements (if there is any) are included as a sublist with less than `n` elements.
If `iL` is empty, then function should return `[]` (will also accept `[[]]`).

The type of `groupNleft` can be one of the following:
```
groupNleft :: Int -> [a] -> [[a]]
groupNleft :: (Num t) => t -> [a] -> [[a]]
```

Note: this function is not required to be tail-recursive.

Examples:

```
> groupNleft 3 [1, 2, 3, 4, 5, 6, 7, 8]
[[1,2,3],[4,5,6],[7,8]]

> groupNleft 5 [1, 2, 3, 4, 5, 6, 7, 8]
[[1,2,3,4,5],[6,7,8]]

> groupNleft 2 [(1,"a"),(2,"b"),(3,"c"),(4,"d"),(5,"e"),(6,"f")]
[[(1,"a"),(2,"b")],[(3,"c"),(4,"d")],[(5,"e"),(6,"f")]]

> groupNleft 2 [1, 2, 3, 4, 5, 6, 7, 8]
[[1,2],[3,4],[5,6],[7,8]]

> groupNleft 3 []
[]       -- will also accept [[]]
```

Testing your functions

We will be using the `HUnit` unit testing package in CptS355. See http://hackage.haskell.org/package/HUnit for additional documentation.

You may install `HUnit` in 3 different ways:

Option1 (using cabal installer) :

Run the following commands on the terminal.

```
cabal update
cabal v2-install --lib HUnit
```

Option2 (from source) :

1. First install `call-stack` package
   - Download the package from here: http://hackage.haskell.org/package/call-stack . Download the `call-stack-0.2.0.tar.gz` file and extract it.

   (To extract .gz and .tar files on Windows you may use 7zip (https://www.7-zip.org/))
   - Then switch to the `call-stack` directory and install `call-stack` using the following commands at the terminal/command prompt:

     ```
     runhaskell Setup configure
     runhaskell Setup build
     runhaskell Setup install
     ```
   - If you get "permission denied" errors:
     - on Windows, run the terminal or command line as administrator.
     - on Mac and Linux, run the command in `'sudo'` mode (or login as root).
2. Next install `HUnit` package:
   - Download the package from here: http://hackage.haskell.org/package/HUnit. Download the `HUnit-1.2.5.2.tar.gz` file and extract it.
   - Then switch to the `HUnit` directory and install `HUnit` using the following commands at the terminal/command prompt: