Section 4: April 19, 2018
fold, types, and list comprehension

Q1: (Q2 from mini-exercises 2)
Write a function `concat'` that concatenates a list of lists. Use `foldr`. (There is a function `concat` in the Prelude that does this, hence the different name.)

Q2: Types

a) Write a Haskell type `Point` that represents a point in 2D space with 2 doubles (x and y)

b) Write a Haskell function `distanceBetween` to calculate the distance between two points.

c) Write a Haskell function `shiftPoints` that shifts a list of points by a given point (use `map`)

d) Write a Haskell function `totalPath` that returns the total distance between adjacent pairs in a given list of `Points`.

We should be able to create the following points and call the our two functions on them as shown below:

```
x = Point 3 4
y = Point 5 12
z = Point 6 8
d = distanceBetween x y
shifted = shiftPoints [x,y] z
d’ = totalPath [x,y,z]
```

Q3: Types

Suppose that we have the following definition of the member function in Haskell:

```
member x [] = False
member x (y:ys)
       | x==y = True
       | otherwise = member x ys
```

What is the type of ==? (Try `:t (==)`)

Circle each type declaration that is a correct type for member. (Not necessarily the most general type, just a correct one.)

A. `member :: a -> [a] -> Bool`
B. `member :: Bool -> Bool -> Bool`
C. `member :: [Integer] -> [Integer] -> Bool`
D. `member :: (Eq a) => [a] -> [[a]] -> Bool`
E. `member :: (Ord a) => a -> [a] -> Bool`
F. `member :: (Eq a) => a -> [a] -> Bool`
G. `member :: [Char]-> [[Char]] -> Bool`

Which of the above types, if any, is the most general type for member?

Q4: List comprehension

Write the haskell code to bind the following lists to the variables x and y (respectively)
(Challenge: Try to think of multiple ways of doing each binding)

a) Bind the following list to the variable x: `[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30]`

b) Bind the following list to the variable y:
`[-1,2,-3,4,-5,6,-7,8,-9,10,-11,12,-13,14,-15,16,-17,18,-19,20]`

Q5: #tbt Tail Recursion and foldr

Write a tail recursive haskell method to compute the average of a list of numbers (the average of an empty list can be 0).

Now write the same method, but use a helper called `sumCount` that uses `foldr` to return an Integer pair (with the first number being the sum of the list, and the second being the count).