csci54 – discrete math & functional programming review

- lists, tuples
- types
- syntax:
  - list comprehensions
  - pattern matching
  - guards
  - where
  - let-in
- higher-order functions
  - map, filter
  - foldr, foldl
  - anonymous functions

Recursion throughout

Group review suggestions

- **type system, type signatures (x4)**
  - types vs. type classes
  - type classes, "main type classes" (x2)
  - when to use Num, Integral, Int, Integer, etc
  - fromIntegral
  - intuition for how Haskell derives type signature
- **higher order functions (x3)**
  - foldr vs. foldl (x2)
  - filter (especially with multiple conditions)
  - types of higher order functions (x2)
- **curried functions (x2)**
  - example of "only one parameter" in function that takes multiple parameters
- **specific constructs**
  - guards
  - where
  - let
  - list comprehensions (x2)
  - anonymous functions
- **working with lists**
  - difference between list recursion and pattern matching
  - examples where things are taken off of the end of the list, such as using init
  - tail vs. last
- question 2 part 2

types and type classes

- examples of types:
  - Bool, Int, Integer, Char, String, Float, Rational
  - [Bool], (Int, Char), ([[Float]], (Int,Int), [Char])

- examples of type classes:
  - Num, Integral
  - Eq, Ord, Enum

```
f _ [] = []
f y (x:xs) = [y..x] ++ xs

g [] = ""
g (x:xs) = let z = xs ++ "s" in (g xs) ++ z

h _ [] = []
h b (x:xs)
   | b = x:(h False xs)
   | otherwise = h True xs

j x = [(a,b) | a <- [1..x], b <- [(-1),(-2)..(-5)], b * b == a]
```

exists

- Write a function `exists :: (a -> Bool) -> [a] -> Bool` which takes a predicate and a list and returns True if and only if at least one element in the list satisfies the predicate.
  - in a way that uses pattern matching? guards?
  - in a way that uses foldr? foldl?
  - in a way that uses anonymous functions?
  - in a way that uses a filter and/or map?

greaterThan

- Write a function `exists :: (a -> Bool) -> [a] -> Bool` which takes a predicate and a list and returns True if and only if at least one element in the list satisfies the predicate.

- How would you use `exists` to write a function `greaterThan` that takes an element and a list and returns True if any element in the list is larger than the given element?

```
greaterThan :: Ord a => a -> [a] -> Bool
```

folds

- What do the following evaluate to?

```
foldr (-) 0 [8,7,6,5]

foldl (-) 0 [8,7,6,5]
```

- Use `foldr` to define a function `sumSquares` which takes an integer n as its argument and returns the sum of the squares of the integers from 1 to n. Do this with and without `map`