In-Class Worksheet

Discrete Math & Functional Programming— CSCI 054— Spring 2024

Instructor: Osborn

What are the types of the following functions?

```
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]
```

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.

- Using pattern matching? Guards?
- Using foldr/foldl?
- Using filter/map?

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

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

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 both with and without `map`.