Haskell Hodgepodge part 2

- `map`, `filter`, `all`
- Extended example

map

```
map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (x : l) = f x : map f l
```

Recall that `(+ 1) :: Int -> Int` adds `1` to its input. Equationally computing:

```
map (+ 1) [1,2,3]
= map (+ 1) (1 : [2,3])
= (+1) 1 : map (+ 1) [2,3]
= (+1) 1 : (+ 1) 2 : map (+ 1) [3]
= (+1) 1 : (+ 1) 2 : (+ 1) 3 : map (+ 1) []
= (+1) 1 : (+ 1) 2 : (+ 1) 3 : []
= [(+1) 1, (+ 1) 2, (+ 1) 3]
= [2, 3, 4]
```

filter

```
filter :: (a -> Bool) -> [a] -> [a]
filter p [] = []
filter p (x:l) | p x = x : filter p l
filter p (x:l) = x : filter p l
```

(<= 3) is True if input ≤ 3. Computing:

```
filter (<= 3) [4,3,0]
= filter (<= 3) [3,0] =
= 3 : filter (<= 3) [0]
= 3 : 0 : filter (<= 3) []
= 3 : 0 : []
= [3,0]
```

all

```haskell
all :: (a -> Bool) -> [a] -> [a]
all p [] = True
all p (x : l)| p x = all p l
all _ _ = False
```

Exercise: equationally compute

```haskell
all (<= 3) [0, 3, 2]
```