csci54 – discrete math & functional programming
higher order functions

"Haskell functions can take functions as parameters and return functions as return values. A function that does either of those is called a higher order function. Higher order functions aren’t just a part of the Haskell experience, they pretty much are the Haskell experience."

map and filter (from last time)

- `map :: (a -> b) -> [a] -> [b]`
  - takes a function that maps elements of type a to type b
  - applies the function to every element in a list of type a and returns a list of the results (which have type b)

```
ghci> map length ["ab", "aaaaa", "b"]
ghci> map (^3) [1,3,6]
```

- `filter :: (a -> Bool) -> [a] -> [a]`
  - takes a function that maps elements of type a to True/False (a predicate)
  - applies the function to every element in a list of type a and returns only those elements for which the function returns True

```
ghci> headA x = (head x) == 'a'
ghci> filter headA ["ab", "aaaaa", "b"]
```

Curried functions

- Every function in Haskell only takes one parameter (!!)
- What does that mean?

```
ghci> mult x y z = x * y * z
```

```
ghci> mult x y z = x * y * z
ghci> let mult10 = mult 2 5 in map mult10 [1,2,3]
```

Practice

- Write a function `multFirst :: [Integer] -> [Integer]` which returns a list containing the products of the first and n'th elements of the input.
  - For example: `multFirst [2,3,4,5] = [6,8,10]`

- Does your function use a higher-order function? If not, how could you write it using a higher order function?

map and filter

- `map :: (a -> b) -> [a] -> [b]`
  - takes a function that maps elements of type a to type b
  - applies the function to every element in a list of type a and returns a list of the results (which have type b)

- `filter :: (a -> Bool) -> [a] -> [a]`
  - takes a function that maps elements of type a to True/False (a predicate)
  - applies the function to every element in a list of type a and returns only those elements for which the function returns True

- how would you implement `map`? `filter`?

practice problem

- The `mapish` function takes a list of functions and a single element x. It then returns a list of the results of applying each function to x. Implement the `mapish` function.

```
ghci> mapish [(+1), (*3)] 10
[11, 30]
```

- what is the type of the `mapish` function?

What if you wanted to mapish:

$f1(x) = x^2 + 1$

$f2(x) = 4x-10$

lambdas (aka anonymous functions)

- functions that don't have names
- functions that you use once in the context of some other function

```
ghci> headA x = (head x) == 'a'
ghci> filter headA ["ab", "aaaaa", "b"]
```

```
ghci> filter (\y -> (head y) == 'a') ["ab", "aaaaa", "b"]
```

- syntax: `\a b -> (a * b + 10)`
  - starts with \ (meant to resemble λ).
  - -> separates parameters from what the function evaluates to