csci54 – discrete math & functional programming
lambdas and folds
Studying for the checkpoint (a week from today)

- Checkpoint will be closed everything (including, for example, calculators) **except** you can have one double-sided 8.5"x11" page of notes and something to write with (pen, pencil).

- Suggestions for reviewing
  - Look over material
    - topics from lecture/problem sets; more detail in textbook
    - do problems on the material: worksheets, group assignments, problem set problems
  - Practice writing down solutions on paper in full
    - duplicate the feeling of seeing a completely new problem and having to write something down
    - come up with variations of problems you've seen (e.g. different problems or solving the same problem in different way)

- Will post study guide later today
practice problem from last time

- 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`.

\[
\text{ghci}\geq \text{mapish } [(+1), (*3)] \ 10
\quad \quad [11, 30]
\]

- Implement the `mapish` function. What is the type of the `mapish` function?

\[
\text{mapish} :: [\text{a} -> \text{b}] \to \text{a} \to [\text{b}]
\]
\[
\text{mapish} \ [\] \_ = []
\]
\[
\text{mapish} \ (f:fs) \ x = (f \ x) : (\text{mapish} \ fs \ x)
\]

use `mapish` to implement a function `f` that takes a number `x` and computes:

\[
\text{f1}(x) = x^2 + 1
\]
\[
\text{f2}(x) = 4x - 10
\]
lambdas (aka anonymous functions)

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

```ghci
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
lambdas (aka anonymous functions)

- note that if we wanted a function `headA` such that it would take out the elements that started with the character 'A', we could define it as follows:

  ```ghci
  headA = filter (\y -> (head y) == 'A')
  ```

- practice: what is the type of the function `foo`? what does it do?

  ```
  foo y zs = map (\x -> x^y) zs
  ```
One more built-in higher order function

- map, filter, reduce

How would you write a function `sumList` that returned the sum of a list of integers?  
How would you write a function `prodList` that returned the product of a list of integers?

```haskell
sumList [] = 0
sumList (x:xs) = x + (sumList xs)

prodList [] = 1
prodList (x:xs) = x * (prodList xs)
```

- what is similar?
- what is different?

In Haskell "reduce" is referred to as "fold"
foldr' :: (b -> b -> b) -> b -> [b] -> b
Right fold (foldr)

foldr' :: (b -> b -> b) -> b -> [b] -> b

- foldr (+) 0 [3,2,6]
  - very, very informally can think:
    - [3,2,6] is really 3:2:6:[].
    - Replace [] with the base case 0 (sometimes called “seed” value)
    - Replace : with the operator (+)
  - associate to the right
  - 3 + (2 + (6 + 0))

- how would you write sumList and prodList using foldr?
foldr and foldl

foldr' :: (b -> b -> b) -> b -> [b] -> b

- foldr (+) 0 [3,2,6]
  - informally can think of as: [3,2,6] is really 3:2:6:[]. Replace [] with the base case and the : with the operator
  - associate to the right
  - 3 + (2 + (6 + 0))

- foldl - same idea but associates to the left
foldr and foldl

\[
\text{foldr} f x [y_1, y_2, \ldots, y_k] = f y_1 (f y_2 (\ldots (f y_k x) \ldots))
\]

\[
\text{foldl'} :: (a \rightarrow a \rightarrow a) \rightarrow a \rightarrow [a] \rightarrow a
\]

\[
\text{foldl} f x [y_1, y_2, \ldots, y_k] = f (\ldots (f (f x y_1) y_2) \ldots) y_k
\]

\[
\text{foldr} (+) 0 [3, 2, 6]
\]
\[
\text{foldl} (+) 0 [3, 2, 6]
\]
practice with folds

\[
\text{foldr } f \ x \ [y_1, y_2, \ldots, y_k] = f \ y_1 \ (f \ y_2 \ (\ldots \ (f \ y_k \ x) \ \ldots)) \\
\text{foldl } f \ x \ [y_1, y_2, \ldots, y_k] = f \ (\ldots \ (f \ (f \ x \ y_1) \ y_2) \ \ldots) \ y_k
\]

- The following evaluate to two different values:
  - \(\text{foldr } (^) \ 1 \ [2,3]\)
  - \(\text{foldl } (^) \ 1 \ [2,3]\)

- What do they evaluate to and why?
and a hint of something more . . .

- \( \text{foldr } f x [y_1, y_2, \ldots, y_k] = f y_1 (f y_2 (\ldots (f y_k x) \ldots)) \)

- what does the following do?

\[
\text{foldr } (\_ \_ s -> 1 + s) \_ \_ 0 \text{ "abcde"}
\]

- what does this tell you about the type signature?

\[
\text{foldr'' :: } (a -> b -> b) -> b -> [a] -> b
\]

- (but really it's this:

\[
\text{foldr :: Foldable t => } (a -> b -> b) -> b -> t a -> b
\]