csci54 – discrete math & functional programming
higher order functions

Last time

- reviewed types and talked about inferring the type of a function

- Haskell constructs:
  - pattern matching
  - guards
  - where
  - let

  - helper functions

```
import Data.Char

mystery'' :: [Char] -> [Char] -> String
mystery'' x y
  | secondHalf aL && secondHalf bL = "group 4"
  | secondHalf aL && not (secondHalf bL) = "group 3"
  | not (secondHalf aL) && secondHalf bL = "group 2"
  | otherwise = "group 1"
  where (a:_) = x
        (b:_) = y
        aL = toLower a
        bL = toLower b
        secondHalf c = if (c > 'm')
                       then True
                       else False
```

pattern matching can be used in many ways

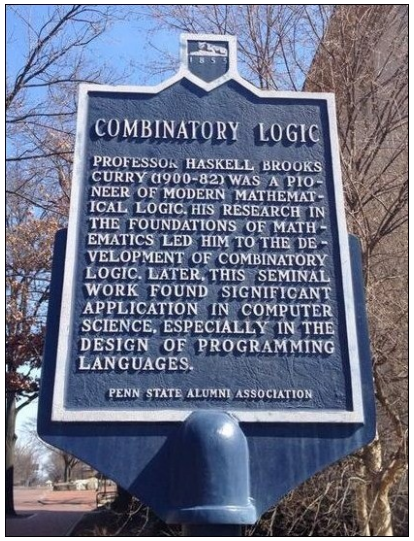
```
secondHalf c = (c > 'm')
```

Practice question

- Consider a function `every0ther` that takes a list and returns a new list consisting of every other element in the original list starting with the first element. As an example, `every0ther [1,5,2,4,-1]` should return `[1,2,-1]`
  - What is the type of `every0ther`?
  - How would you implement `every0ther` using pattern matching?

Some history

Haskell B. Curry



Combinatory Logic - Professor Haskell Brooks Curry (1900-82) was a pioneer of modern mathematical logic. His research in the foundations of mathematics led him to the development of combinatory logic. Later, this influential work found significant application in computer science, especially in the design of programming languages.

Some very useful built-in Haskell functions

- map and filter

```
ghci> isEven x = (x `mod` 2 == 0)
ghci> map isEven [1..10]
ghci> filter isEven [1..10]
```

- examples

```
ghci> map head ["abc", "def", "AAA"]
ghci> let l x = (length x > 3) in filter l ["apple", "pen", "banana", "x"]
```

- what is the type of the map function?

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."

- What's the type of the map function?

```
map :: (a -> b) -> [a] -> [b]
```

- What's the type of the filter function?
-

```
filter :: (a -> Bool) -> [a] -> [a]
```

Exploring higher-order functions

- Are these type signatures different? Could they all describe the same function?

```
f1 :: (a -> a) -> a
f2 :: a -> (a -> a)
f3 :: a -> a -> a
```

- consider the following function

```
ghci> isDiv x y = y `mod` x == 0
```

  - what is the type of `isDiv`?
  - what if you wanted to use `isDiv` with filter, say to select elements divisible by 7?

Exploring higher-order functions

- consider:

```
ghci> isDiv x y = y `mod` x == 0
ghci> head (filter (isDiv 7) [1000,999,..1])
```

- what is the type of `isDiv`?

```
isDiv :: Integral a => a -> a -> Bool
```

- what is the type of `(isDiv 7)`?

```
(isDiv 7) :: Integral a => a -> Bool
```

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]
```

"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."

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?