CS344 - Higher Order Functions

Note Title 3/21/2012

- HW due Monday in class
  - Next HW still due next Friday

Where Bindings

We can set local names for expressions.

Ex:

```
initals :: String -> String -> String
initals firstname lastname =
  [f] ++ ". " ++ [l] ++ "."
  where (f:_) = firstname
        (l:_) = lastname
```

Let bindings

let <bindings> in <expression>

> let square x = x * x in (square 5, square 3, square 2)

↳ (25, 9, 4)

Difference:
lets are expressions themselves!
(where just gives a local nickname)

Ex: > 4 * (let a = 9 in a+1) + 2
↳ 42

Ex: function to compute area of a cylinder

Recursion in Haskell

How do we find the maximum element in a list?

Sequential (or linear) search

```
maximum' []     = error
maximum' [x]    = x
maximum' (x:xs) | x > maxTail = x
                | otherwise   = maxTail
  where maxTail = maximum' xs
```

How do we do it recursively?

max (list) ⟶ returns max element

```
maximum' (x:xs) = max x (maximum' xs)
```

How about reversing a list?

Code this one yourself, & think recursively!

Start:

rev :: [a] → [a]

Higher Order Functions

Haskell functions can take other functions as input parameters.

In fact, we've been doing this already.

Why? Haskell functions can only have one input parameter!

```
(max 4) 5
```

↑ Space is a function application (& has highest precedence)

So: max 4 returns a function which returns either 4 or the input parameter

Really, have (max 4) 5 -

Check: >:t max

↳ (Ord a) => a -> (a -> a)

return type

Some simple applications

Look at multThree.hs

↳ simple function to multiply 3 #s together.

```
> let multTwoWith6 = multThree 6
> let multWith12 = multTwoWith6 2
```

Higher Order functions (cont)

So by calling a function w/ not enough parameters, we're creating functions as we go.

Note: try without a let

```
>multThree 3 4
```

Produces a function, which isn't part of the Show type class, so Haskell can't print it.

Infix functions : use ( )

Ex:

```
divideByTen :: (Floating a) => a -> a
divideByTen = (/10)
```

```
divideByTen 200
```

↳ 20

Ex:

```
isUpper :: Char -> Bool
isUpper = (`elem` ['A'..'Z'])
```

Now: functions as parameters

applyTwice example:

Note: -parenthesis in type are now mandatory! Why?

Ex: > applyTwice (+3) 10

> applyTwice ("HAHA "++) "HEY"

> applyTwice (multThree 2 2) 10

careful! ⇝ > applyTwice (3:) [1]