CS 344 - Back to Haskell

Note Title 3/9/2012

Announcements

- HW 6 is up due Friday after break
- Today is in-class work day

Haskell

Purely functional — No variables.

- strongly typed (uses inference)

Functions (late 2)

Example

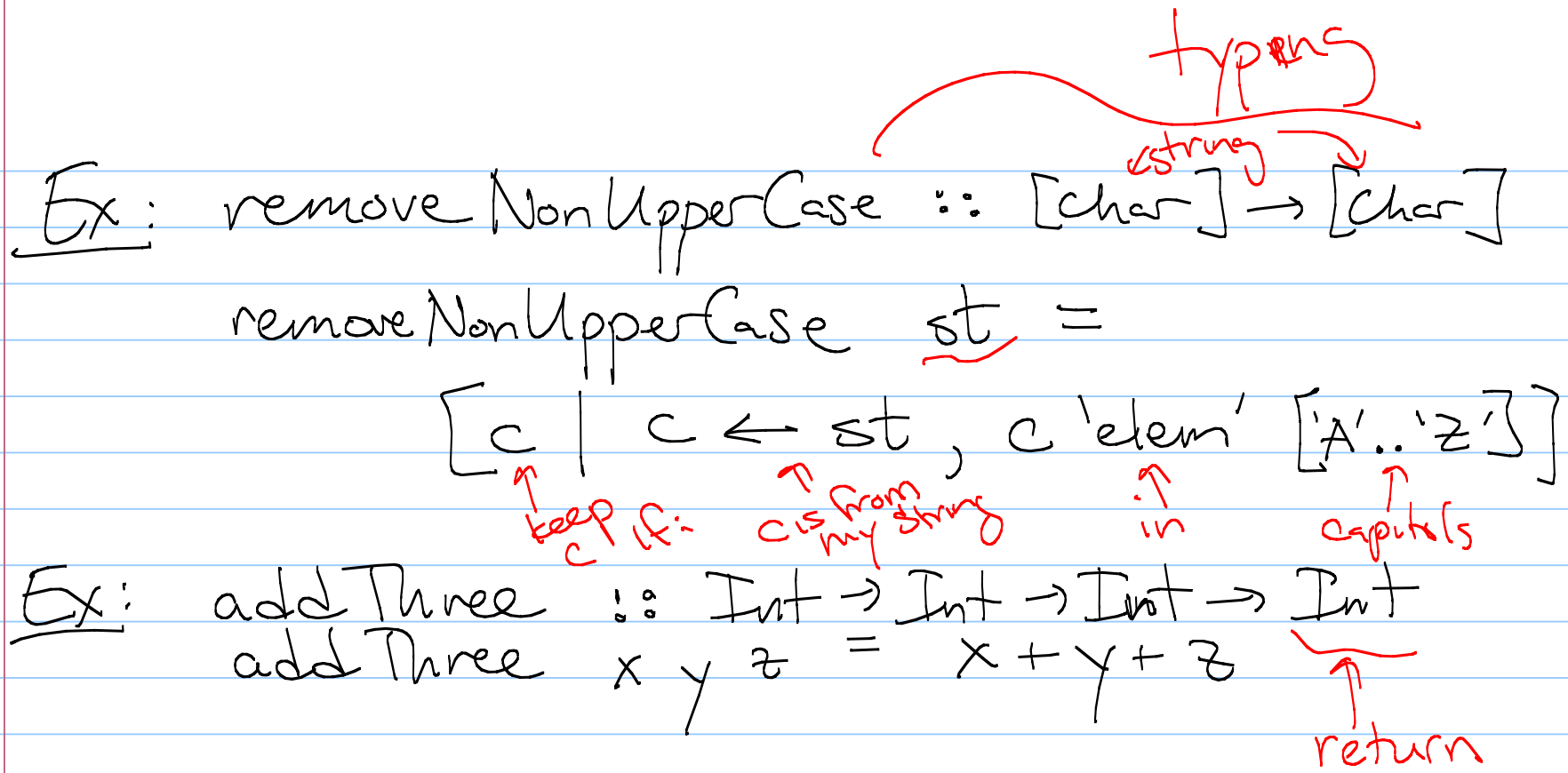
Notes: It is considered good practice to type your functions!

```
functionname :: type -> type
```

type could be list, too —

go check:

```
:t "Hello"
:t 'a'
:t (True, 'a')
:t ('a','b','c')
```


Ex: removeNonUpperCase :: [char] → [char]
typing
[string]
removeNonUpperCase st =
[c | c ← st, c 'elem' ['A'..'Z']]
keep if:
c is from my string
in
capitals
Ex: addThree :: Int → Int → Int → Int
addThree x y z = x + y + z
return

Nice Feature:

If you don't know the type, you can write your function and then check it!

Type Variables

Try :t head

(What is head again?)

↳ returns first element

Haskell allows for polymorphism in this way

Try fst - takes tuple & returns first element

Exercise

Get into groups (preferably w/ at least 1 computer per group).

Write the fibonacci function in Haskell. (& Please type it).

Note: Submit to me w/ names of group for credit!

```
fib :: Integer -> Integer
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Type Classes

Not an O-O class!

An interface that defines some behavior.
(More like Java interfaces.)

:t (==)

What is ⇒?
a class constraint

Type classes

This means that the type of the two values (a) must be a member of the Eq class.

Num: numeric types

Eq: for types that support equality testing

Ord: types that have an ordering

Show: types that can be presented as strings

Read: ↗ (opposite)

Strong pattern Matching

Can use pattern matching in unexpected ways.

addVectors example

Another:

```
> let mylist = [(1,3),(4,3),(5,6)]
> [a+b | (a,b) <- mylist]
```

Our own head function

```
head' :: [a] -> a
head' []    = error "Can't call head on empty list"
head' (x:_) = x
```

See firsttwo.hs