CS212: Paradigms, Spring 2016

Dr. Joshua B. Gross, Blackburn College

Haskell is Good For You

Some notes for Friday, along with a test file (recusion-examples.hs in the haskell directory in the lecture repo) since we don't have a textbook (although use the tutorials!).

Think of parts 4 & 5 as the prep questions and exercises in your Java book. Except maybe at least consider doing these. I think it would take at most 1-2 hours to complete all of this, and most of it can be done in one.

1. Things to Remember

- It's implied that anything created in a function will be returned. It's not always true, but remember that the idea is that you pass in some value and the function passing you back some value. Usually not the same one.

- Still, **always** be clear what a function takes in and what it outputs. If you have a function that takes in a number and creates a list, state that. Even if it's obvious. Even if the name of the function is takeInANumberAndReturnAList.

- If we have two functions that do the same thing the same way, then we can call them foo and foo' (spoken "foo prime"). Or foo" (spoken "foo prime prime"). Is this a good idea? Well…

- There are no variables in the sense that one label in one context can have its contents changed, and no content can be changed.
    - You can't add an element to a list; you can create a new list with an added element
    - You can't change the content of a list; you can create a copy with different values
    - You can't add to (or subtract from, etc.) a variable; you create a new number

2. Some Useful Stuff

How do you get to your code?

- If you are using Cygwin (note you use / to separate folders):
    - cd /cygdrive/[whatever drive letter]/[path to directory where your code is]
    - ghcii.sh
    - :l filenamewithoutdothsextension
    - :r

      :r reloads everything, so if you created new labels/functions with let, they're gone
- If you are using command (aka cmd, aka DOS; note you use \ to separate folders):

- cd [drive letter]:\[path to directory where your code is]
- ghci
- :l filenamewithoutdothsextension
- :r

  :r reloads everything, so if you created new labels/functions with let, they're gone

- Elements and lists:

consing **[and what's the reverse? ++?]**
x vs (x:xs)
using $ (only in debugging)

String manipulation?

Java limit on recursion - no point usually, much slower, not really functional

3. Types

Haskell likes type signatures, which look like:

```
length  :: Foldable t => t a -> Int
length' :: [a] -> Integer
print   :: Show a => a -> IO ()
```

We don't want (or need) to deal with types, so we can do either of a couple of things:

1. When starting up ghci, set the ImplicitParams attribute:
   ```
   ghcii.sh -XImplicitParams
   ```

2. Include the ImplicitParams pragma (an instruction to the compiler/interpreter) in the file:
   ```
   {-# LANGUAGE ImplicitParams #-}
   ```

3. Don't try to do anything wild with types, like converting from Char to Int. Or Int to Integer. Or (please no) Int to [Integer]. You won't need this.

4. Playing with Recursion Examples

1. Load the recursion-examples.hs file into ghci

2. Try each function (including the label cl)
   a. Try each function with no argument
   b. Try each function with the correct type of argument (e.g., a number instead of a list)

    c. Try each function with an incorrect type of argument (e.g., a list instead of a number)

3. How fast is Haskell? This is *interpreted* Haskell, which means it's slow. If you compiled it, it would be much faster.
    a. How long does it take to add every number 1,000,000 to 1?
    b. Does it take Haskell longer to compute a long answer, or display a long answer?

4. There are two issues:
    a. A bug
    b. A "will it work?" question

   Fix the first, answer the second

5. Try calling one function inside of a call to another function
    a. Add 10 to each element of cl
    b. Add 11 to each element of a list [10, 9, 8, 7…]
    c. Add 10 to list cl, printing out half of the operations
    d. Create a new label (cl') that contains the content from above
    e. Print cl'
    f. Create a new label (cl'') that contains the content from above with 5 added to it
    g. Print cl''; did it do what you expected?

5. Write Some Recursion Examples

- multiplyBy5 - take a list and return a new list with each element multiplied by 5
- multiplyByN - take a list and a number and return a new list with each element multiplied by the number
- product (take a list, return each element times the next, etc.)
- embed - take a list, put each element inside its own list, and add each of those lists to a new list
- sublist - take a list, a start index, and a stop index, and return a new list containing only the elements between the start and stop index

6. Maybe Try to Think of Your Own?