CS212: Paradigms

Dr. Gross, Spring 2016

This lab has no home portion.

Warning: you may (will) get these errors:

```
<interactive>:6:12: lexical error at character '\SYN'
<interactive>:21:9: lexical error at character '\ESC'
```

Ghci is very sensitive to character input. Try again and **don't copy & paste**.

Part 1

First, let's get to know ghci:

Start the DOS command line (go to Start, type cmd, hit return) and type ghci

OR

Start Cygwin64 Terminal (go to Start and type cyg, and select the Cygwin64 Terminal) and type ghcii.sh

Let's do some math:

```
        5 + 5
        5 - 5
        5 * 5
        5 / 5
```

Wait, that works? But I said… the answer is that it works sometimes. Instead use the code below.

```
        5 `div` 5
```

There's a difference between this one and the prior one - what is it? If you can't find it, ask.

Wow, I'm sure that was overwhelming. When you've somehow managed to calm down, let's get to some lists. Can't do anything without lists.

```
        [1, 3, 5, 7, 9]
        head [1, 3, 5, 7, 9]
        tail [1, 3, 5, 7, 9]
        init [1, 3, 5, 7, 9]
        last [1, 3, 5, 7, 9]
        [[1,2] 3, 4, 5]
```

Yep, this is what I tried to do on Wednesday (unintentionally). Every element of a list must be of the same type. So why doesn't this work?

```
[[1,2], [3], [4, 5, 6]]
```

There's gotta be an easier way than typing this over and over (or copying and pasting)

```
        let x = [4, 7, 9, 6, 3, 2, 1]
        let y = [[1,2], [3], [4, 5, 6]]
```

Then run the following commands (expressions), but to make this fun, write down what you think they will do *before you run them*. I know, I'm a cruel taskmaster.

```
        x
        y
        head x
        head y
```

```
head head x
head head y
head (head x)
head (head y)
head (head (head x))
head (head (head y))

tail x
init x
last x
reverse x

tail x
init x
last x
reverse x

null [1]
null []
null [[]]
null
null 0
```

Is there something like JavaDoc API? Well, sort of:
https://hackage.haskell.org/package/base-4.8.2.0
It's all there. Just some of it… well, it won't necessarily be clear. Just as the Java API wasn't before, and still isn't sometimes. The packages you need to/want to know are Data.List and Prelude. You won't be expected to know anything you can't possible or understand. Also, just because you have the definition doesn't mean you have an example. Google for examples

Part 2

OK, so what exactly can we do with lists?

```
sum x
length x
length x
(sum x) `div` (length x)
```

What does this do?

```
(sum y) `div` (length y)
```

What does this do? Why doesn't it work?

Part 3

Making functions. Ok, Haskell is a functional language, so how do we make functions? Let's use llet.

```
let absolute x = if (x <0) then -x else x
```

In this example, absolute is our function name, and x is our parameter or argument. The rest of the code is pretty obvious, but a couple of notes: first, note the word "then", and second, *you always have to have an else.*

```
absolute 5
absolute -5
```

What does that error mean? This is one of those annoying things that tutorials don't tell you about Haskell. The answer why it does this is long, boring, and irrelevant for our purposes. Instead, use:

```
        absolute (-5)
```

Now's probably a good a time as any to introduce how to write Haskell code into files and read those files.

Fire up Notepad++ and save a file called lab10functions.hs to C:\Users\[whatever your username is]. Now we can add code. We don't use let here; we just write our functions.
How about our averaging function above? That's easy to do. Just add:

```
        average x = (sum x) `div` (length x)
```

Note that we don't use let in files.

Now we have to load this into Haskell. We do that with the :load command, or just :l

```
:l myfunctions
```

If you got no errors, let's proceed.

```
        average [10]
```

Wait, that worked? Why?

```
        average [1, 2, 3]
        average [0, 1, 2]
        average 10
```

OK, you had to know that wouldn't work. Haskell doesn't compare types until runtime. Java would catch this as a compiler error. What about:

```
        average []
```

Different problem, but it still doesn't work.

Part 4: Your Turn

What kind of function can you make? You know you don't know how to loop, so don't think about that. Make some simple functions. See if you can get them to run.

Part 4': I Can't Think of Anything

Try some of the list and arithmetic problems here:
https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems
You won't be able to do all of these. Some of the solutions won't make sense. It's useful to note that func, func', and func'', etc., are all different functions, so the solutions contain multiple explanations.

Part 5: Before You Go

Make a directory in your homework repository, right off the root (firstname.lastname) called haskell. Crazy with these naming schemes, right? Then move/copy your code into that directory, open up Netbeans, add, and commit.

Grading Rubric

Did you show up? Then you get credit. :)
Did you not show up? Then you did not get credit. :(